4-19-2016

Solar Tracking and Lighting Design for a Multimedia Sculpture

Alexandra Ferguson
Western Michigan University, aferguson2@gmail.com

Follow this and additional works at: http://scholarworks.wmich.edu/honors_theses
Part of the Electrical and Electronics Commons, and the Power and Energy Commons

Recommended Citation
Solar Tracking and Lighting Design
for a Multimedia Sculpture
Final Report

April 25th, 2016
Alexandra Ferguson
Cody Middleton
Chelsea Russell
ABSTRACT

A lighting system utilizing solar energy was built to illuminate a wireframe multimedia sculpture. Increased power generation from the solar cells was achieved using a solar tracking mechanism. Color-selectable light emitting diodes were used to provide the user with unlimited customization options. A digital control system was designed using an Arduino® microcontroller to configure the color of the lights and the positioning of the solar panel. The sculpture will create a sense of community and sustainability among students at the school.
DISCLAIMER

This report was generated by a group of engineering seniors at Western Michigan University. It is primarily a record of a project conducted by these students as part of curriculum requirements for being awarded an engineering degree. Western Michigan University makes no representation that the material contained in this report is error free or complete in all respects. Therefore, Western Michigan University, its faculty, its administration or the students make no recommendation for use of said material and take no responsibility for such usage. Thus persons or organizations that choose to use said material for such usage do so at their own risk.
# TABLE OF CONTENTS

ABSTRACT .................................................................................................................................... ii  
TABLE OF CONTENTS ............................................................................................................... iv  
1.0 SUMMARY .............................................................................................................................. 6  
2.0 INTRODUCTION .................................................................................................................... 7  
3.0 SPECIFICATIONS ................................................................................................................. 11  
  3.1 Physical Characteristics.................................................................................................. 11  
    3.1.1 Size .......................................................................................................................... 11  
    3.1.2 Electrical ................................................................................................................. 11  
    3.1.3 Environmental ......................................................................................................... 11  
    3.1.4 Location .................................................................................................................. 11  
  3.2 Functionality................................................................................................................... 11  
    3.2.1 Solar Panel .................................................................................................................... 11  
    3.2.2 Lighting System ...................................................................................................... 11  
    3.2.3 Control Box ............................................................................................................. 11  
4.0 DESIGN .................................................................................................................................. 12  
  4.1 Mechanical .......................................................................................................................... 12  
    4.1.1 Design ........................................................................................................................... 12  
    4.1.2 3D Modeling ................................................................................................................. 14  
    4.1.3 Build ............................................................................................................................. 21  
    4.1.3 Improvements ............................................................................................................... 22  
  4.2 Electrical System ................................................................................................................. 26  
    4.2.1 Motor Control Circuit ................................................................................................... 26  
    4.2.2 Positional Feedback ...................................................................................................... 27  
    4.2.3 LED Control Circuit ..................................................................................................... 29  
    4.2.4 Connections .................................................................................................................. 31  
    4.2.5 Switches ........................................................................................................................ 32  
    4.2.6 Miscellaneous Circuit Components .............................................................................. 34  
    4.2.7 Electronic Build ............................................................................................................ 34  
  4.3 Programming ....................................................................................................................... 36  
    4.3.1 Timer Interrupts ............................................................................................................ 36  
    4.3.2 External Interrupts ........................................................................................................ 36  
    4.3.3 Analog Inputs ............................................................................................................... 36
1.0 SUMMARY

Coloma High School has built a wire frame shaped like a person which they plan to cover with medallions made by student groups at the school. The sculpture is titled “A Coloma Student Is…” The original specifications for this project include lighting up the sculpture with the school colors and allowing for different options for lighting. The sculpture will be separate from the school power system by utilizing solar panels. In order to optimize the amount of power generated by the solar panels, a solar panel tracking system was utilized.

The design is broken up into three main parts: the mechanical design, the electrical system design and the programming design. The mechanical design chosen supports and rotates the solar panel from a central location. It also has three positions for a cotter-less hitch pin to be in to change the front to back angle based on the season. The final build was done with the help of General Motors. Unfortunately, the motor had some slop in the gears which made the solar panel move without feeding the motor. Struts were added to stabilize the panels, but a stronger, more robust motor is recommended to improve the project. The electrical system design involves circuits to control the motor using a full h-bridge motor driver and photovoltaic isolators to isolate motor noise from the electronic circuit. An accelerometer was used to provide positional feedback in terms of the angle of the solar panel. Limit switches in the motor box provide backup for the accelerometer. The LEDs are RGB, meaning their color can be controlled based on the amount of red, green, and blue (RGB). An Arduino® Due microcontroller was used to control the system. The programming involves functions for tracking the sun based on monitoring the solar panel voltage. Another function is for detecting sunset and sunrise to enter and exit a Night Mode function. The Light Controller function uses external interrupts to change the color and function of the lights based on three user input switches. The lights can be Coloma Green and Coloma Gold; red, white and blue; or rainbow. They can also go through three different functions: solid, where the lights stay a solid color; blink, where they blink on and off; and fade, where they fade from high intensity to low intensity. Finally a Remote Mode allows the user to interface with the system and view diagnostics over the Arduino® Serial Monitor.

Due to funding issues, the full lights will not be purchased and the sculpture will not be installed right away, which caused many original specifications to be altered. Overall this system met all specifications that were not altered over the course of the project. Testing of the solar panel output was done at sunrise and sunset to validate the system. Recommendations include purchasing a more robust motor and using printed circuit boards instead of soldering to prototyping boards. Many future projects could branch off of this, including optimizing the way the solar panel tracks the sun as far as the number of times it tracks per day and how far it should move each time it tracks.
2.0 INTRODUCTION

Coloma High School is a public school located in Coloma, Michigan that is centered on the development of their students through educational and extracurricular activities. The high school has begun the development of a sculpture design concept to be located at the front of their building. The project idea was developed by a former student who completed a wire frame of the sculpture in 2014. This sculpture, titled “A Coloma student is…”, is meant to encompass all of the qualities a student at the high school will develop throughout their tenure.

Coloma High School needed a control system design utilizing solar panels to power their new multimedia sculpture. The central idea was that the sculpture would be constantly changing with something different every time it is looked at. This was accomplished with the addition of multiple lighting options as well as the future addition of medallions representing changing classes and students.

In deciding on lighting options, many types were considered. Light emitting diodes are long lasting and provide the directionality and controllability desired in outdoor lighting [1]. Lighting system that are exposed to the elements must also protect from contact with voltage and current through its grounding system. The system can either be grounded independent of the supply ground point or to the system point, but every fault loop must be analyzed [2]. The environmental impact of the lighting is a hot topic in an evolving society working towards a more environmentally friendly future. Since six percent of greenhouse gases are the result of lighting, the impact of the lights need to be examined [3]. Smart lighting can be considered to eliminate the light usage when it is not necessary. Smart lighting can include turning down or off the light based on weather and activity conditions [3]. These functions can be controlled by sensors and a controller.
Since this project is related to solar energy generation, it is important to note that in Michigan, solar power is subject to specific rules and recommendations [5]. In doing research, deep cycle batteries are most commonly used for solar power because of the way they function. Wet cell (car) batteries are used when short quick bursts of energy are needed and discharging and charging happens very fast, whereas deep cycle batteries have very slow discharges. This is better for solar panels because it goes all day being charged and all night discharging, much longer cycles than a car battery [6]. In some solar panel tracking systems [7,8], DC motors are used. Specifically, AC motors are not typically used because of their “difficulty to control at slow speeds” [9]. The choice of a brushless dc motor was made given their long lifespan and they have a low “total cost of ownership” (TCO) [9]. In order to rotate a large heavy object slowly, a reducing motor is often used [10].

In designing the tracking algorithm, other algorithms were investigated. A system can be used to maximize the amount of solar energy generated from a solar panel. The ‘perturbation and observation’ method maximizes the amount of solar energy generated from a solar panel [11]. This method adjusts the angle of the solar panel and measures the change in output power. If the adjusted angle results in a positive change in power, the panel continues to move in that direction until the change in power becomes negative [11]. Many aspects of this type of tracking were considered when creating the tracking algorithm.

Light emitting diodes or LEDs can be used in lighting applications. Combinational red, green, and blue LEDs can be adjusted using a microcontroller [12]. Color is composed of three elements, hue, saturation, and brightness [12]. One can produce a variety of colors based on the concept of additive color mixing. Additive color mixing combines different intensities of red, green, and blue light. Using a microcontroller to regulate PWM duty cycles on a RGB LED, one
can generate any color [12]. The LED circuit used in this project uses this method of lighting control.

A DC servomotor was investigated for positioning. The angle of the servomotor can be regulated using positional controls. Proportional Integral Derivative controllers are relatively easy to tune and are commonly used in the industry to regulate position [13]. One can use a PID controller cascaded with an H-Bridge to set the position of a DC servomotor. Manipulating the coefficients of the PID controller changes the speed at which the motor will rotate along with the error in the motor positioning [13]. The PID controller can be tuned to operate slower resulting in a more accurate positioning. Eventually, this was not selected, but a worm-gear motor was chosen. This is due to the fact that worm gear motors can hold something in position without drawing current due to their design.

Many patents have been filed that detail improvements and design ideas for solar panel tracking or any other section of this design project. For example, in [15], the tracking system predicts the position of the sun at different times of the day using a software algorithm that takes into account latitude and longitude of the solar array. It also uses a computer model to determine actuations for the motors in order to position the solar panel so that it is orthogonal to the incoming rays of the sun. It claims that this will be done on a periodic basis during daylight hours. The claims further define the start and stop times as related to the position of the sun and that the panel will be in a ‘parked’ position during the night hours. Many of these ideas were considered when designing the algorithms for this project.

In [16], improvements were made in the conversion of solar energy to electrical and thermal energy. These include the use of an unsealed enclosure, rust resistant materials, and a
glass cover. Additionally, state-of-the-art claims outline a water heating circulation system involving a hot water storage tank, one or more hot water taps, and collecting tubes.

State of the art technological advances made in [17] focus on three modes of operation for solar panel functioning. Mode 1 is direct charging from the solar panel, Mode 2 is charging up a rechargeable battery with the solar panel for later use and Mode 3 is utilizing the rechargeable battery to charge the attached device. While this was not incorporated in this project, it is something that can be considered when improving this project.

Based on this research, design parameters were set. This report describes the mechanical, electrical, and programming involved to design and build this control system entailing solar tracking and LED lighting.
3.0 SPECIFICATIONS

In the following section, the original specifications will be presented. The physical specifications have been broken into size, electrical, environmental, and location. The functionality specifications have been broken into solar panels, lighting system, and control box.

3.1 Physical Characteristics

3.1.1 Size

3.1.1.1 All lighting will be contained within the sculpture which is a wire frame in the shape of a person: height 6.5 ft. width 3 ft.

3.1.1.2 Medallions (approximately 2 inches in diameter) which represent departments and groups at the school will be attached to the wire frame.

3.1.2 Electrical

3.1.2.1 Output power of solar panel (3V, 6V, or 12V) will be converted to charge the battery (12V).

3.1.2.2 Output power of battery (12V) will be delivered to the electronics, sculpture lights, and motor (7V-12V).

3.1.2.3 Lighting must be connected so that replacements or additions of lights can be made easily.

3.1.3 Environmental

3.1.3.1 Sculpture and all exterior components must be weatherproof to be safe from rain, snow and variant temperatures, by utilizing a National Electrical Manufacturers Association Type 4 enclosure [14].

3.1.3.2 All electrical components must be able to withstand extreme Michigan temperature conditions (-20° F / 100° F).

3.1.4 Location

3.1.4.1 Power source will be located at least 10 feet away from the sculpture, not attached to the sculpture or sculpture base.

3.2 Functionality

3.2.1 Solar Panel

3.2.1.1 The solar panel will have controls to track and point in the direction of the highest intensity of the sun.

3.2.1.2 The solar panel will be able to charge the battery sufficiently with no interruption of power for up to seven days of limited sun (clouds).

3.2.2 Lighting System

3.2.2.1 The lights will have the ability to connect to medallions.

3.2.2.2 The lights will be the school colors (green and gold).

3.2.2.3 The lights will be able to change to other colors (i.e. red, white, and blue for Fourth of July).

3.2.2.4 The system will allow for replacing or adding more lighting without rewiring the entire sculpture.

3.2.3 Control Box

3.2.3.1 The system will allow motor control to position solar panel so it is in the greatest intensity of sunlight.

3.2.3.2 The system will convert power from output of solar panel (3V, 6V, or 12V) to battery voltage (12V).

3.2.3.3 The system will convert power from output of battery (12V) in order to power control box, motors, and lights (7V-12V).
4.0 DESIGN

This chapter includes sections on the mechanical design, which includes design, 3D modeling, the final build, and recommended improvements. Then the electrical system is discussed including the motor control circuit, positional feedback, the LED control circuit, the connections, switches, and miscellaneous circuit components. Finally, the programming design and implementation is discussed, including timing and external interrupts, and analog inputs. Then the algorithms are discussed including the tracking algorithm, sunrise and sunset detection, light controller, recovery mode, and remote mode.

4.1 Mechanical

In this section the mechanical design is discussed, including the initial design, 3D modeling, the final system build, and then improvements made on the system.

4.1.1 Design

The mechanical system for this project moved from design to final product within a 12 week period. The goals of the design included 90 degrees of daily rotational movement, 3 seasonal angles, and a durable and inexpensive design. The 90 degrees was chosen so that some portion of the solar panels would be towards the sun at all times. Three seasonal angles were chosen to accommodate for the changing angle of the sun to due south. Spring and fall angles are the same. To begin, several ideas were sketched on a white board in an attempt to optimize the daily rotational angle possible, as seen in Figure 1.

![Figure 1: White Board Sketches](image)

Once a basic design was chosen, a model was built using Kinects®.
The concept seen in Figure 2 was that a motor would rotate a worm gear on the front corner of the system, moving the corner up or down. With even support, this would change the daily angle of the panels. The design also required the movement of two of the legs to change the seasonal angle. The legs were placed into different holes, moving the legs in and out which would move the top of the solar panels up or down. It was found that this design had significant changes in available daily rotational angle depending on the position of the seasonal angle legs. The leg movement also required several rotational parts, reducing the stability of the system.

With the disadvantages of the previous design in mind, Vex® components were used to work through several center based concepts. The use of Vex® components instead of Kinects® allowed us to see the restrictions of a solid system, where Kinects® would allow rotational movement.

Figure 2: Kinects® Model

The design pictured in Figure 3 was the final center based concept. It allowed full rotation in all four directions. It was noted that one of the constraints that would be encountered in a full scale solar panel design would be when the center mechanism would hit the center where the motor needed to be located. This was taken into account when choosing solar panel placement in the 3D modeling designs.
4.1.2 3D Modeling

Design 1

After building a parts list based off of the Vex® model, the first 3D modeled design, seen in Figure 4, was constructed.

Solar Panel Placement

The solar panels were placed on extended bars to allow maximum rotation without hitting the center of the system. One of the extension bars was extended in the opposite direction to allow for the addition of a counter weight. The counterweight was intended to balance out weight of the panels on the system, requiring less torque on the motor.

Mounting Post

A 6x6 wooden post was selected to give the necessary strength to support the system. The post length desired is 8 ft. Three feet of the post should be buried and cemented into the ground to keep the system up right. The additional five feet above ground was determined as an optimal height for a person to be able to reach the system in order to change the seasonal angle.
Motor

The motor chosen for this design, seen in Figure 5, was a TSINY TS-58GZ868-527. This motor was chosen since it was a 12 V, reversible motor with the lowest amperage for the amount of torque required. Required torque was based off of the solar panel distance from the motor shaft and the anticipated weight of snow and gusts of wind. It was found that a compacted cubic foot of snow weighs 20 pounds. Since the panels are constantly moving, it is expected that this will be the maximum amount of snow weight on the panels at any given time. Additionally, wind speeds around Coloma High School average 17.69 mph with storms reaching up to 60 mph. The maximum was selected in torque calculations. However, a significant amount of wind is negligible since the same amount is hitting top and bottom, cancelling out the rotational force.

To hold the motor in place two custom motor mounts were designed to support the motor without limiting motor access or shaft movement.
Aluminum Components

A 25mm solid aluminum shaft was selected to withstand shearing force due to the weight of the solar panels, wind gusts, and snow accumulation. Two 25mm set screw locking pillow bearings were selected to fit the selected shaft size and lock the shaft into place for precise motor movement control. Two 25mm aluminum shaft supports were mounted on the ends of the seasonal shaft to allow for precise perpendicularity and an overall well supported system.

Figure 5: 3D Model Design 1, Daily Rotation

Seasonal Rotation Holes

The seasonal rotation holes seen in Figure 6 allow a pin to slide into holes within a shaft depending on the season. The angle towards true south depends on the latitude location of the panels. Coloma High School is located at a latitude of 42 degrees. Therefore, the summer angle will be 42 degrees towards true south, the spring and fall angles will be 52 degrees, and the winter angle will be 62 degrees. The center hole of the cap is set to the spring and fall angle of 52 degrees with the holes on each side being +/- ten degrees.

Post Cap

The post cap design was a solid sheet of steel that would be bent and welded to fit the 6x6 wooden post. The post would have to be cut to fit the top portion of the cap, and then holes would have to be drilled through the steel cap and wooden post to allow for the seasonal pins into the holes on the aluminum shaft. This shaft would go through the steel cap and wooden post.
Design 2

The system was redesigned due to the donation of two 100W solar panels from General Motors and the desire to remove the counterweight from the system without requiring more torque from the motor.

Solar Panel Placement

The new solar panels added an additional 5 pounds to the system, requiring more torque from the motor or a heavier counter weight. Since torque is proportional to mass and distance, the solar panels needed to be pulled closer to the motor shaft. In order to do this the spacing seen in Figure 7 was created in between the panels for the motor and pillow bearings to sit above the motor shaft axis without hitting the panels during panel movement.
Figure 7: 3D Model Design 2, Full View

**Aluminum Components**

Four additional 25mm aluminum shaft supports were added to the daily rotational shaft to mount the solar panels close to the center of the system and support the panels on both ends of the shaft. Two of these support shafts can been seen in Figure 8.
Final Design

The final build went under several changes from the 3D modeling design to make a stronger, more reliable system.

Aluminum Framing

To mount the panels to the shaft supports, 80/20 aluminum framing was added. This framing, seen in Figure 9, gave additional mounting points on the panels so all of the force was not centralized on four points on the panel.

Post Cap

The post cap was milled out of a solid aluminum block. This guaranteed an exact fit on the post, and removed the need for the post to be cut to fit the top of the cap. In addition to removing the cut wood post, the new cap has water drainage holes in case water ever enters through the pin holes on the top. The combination of these changes greatly reduces the risk of the post rotting due to water damage on an exposed, untreated top section. The holes for the seasonal pin locations were milled into the post cap design, ensuring an accurate degree setting. The milled cap also increased the strength of that part of the system. The cap is mounted to the post by two bolts going all the way through the post and secured with washers and lock nuts.
Motor Box

A watertight box was added to keep the motor and pillow blocks out of the weather. The base of this box can be seen in Figure 10, and the top can be seen in Figure 11. Since the motor was not weatherproof, this was a necessary addition to the system. This also allowed for the addition of limit switches on the motor shaft.

3D Printed Block

A block was 3D printed to verify the angle of the system in the case of disassembly. If the seasonal shaft supports are ever removed, the block can be placed under the motor box on the opposing side of the seasonal holes. With the pin in the middle seasonal hole, the shaft supports can be re-tightened. At these positions, the 52 degree angle is ensured.
4.1.3 Build

The final built system is seen in Figure 11. As previously mentioned, the motor box is placed center to allow the panels to move closer to the motor shaft without the motor box placement being a limitation. The maximum daily rotation was limited to a maximum of 74 degrees to allow for two inches of space between the solar panels and post before it hits the accelerometer coded stop, and one inch between the solar panels and post before it hits the emergency limit switch stop. The control of these stops is discussed in the Programming section.
Inside the motor box, seen in Figure 12, are the two pillow bearings, the motor, and the limit switches. The limit switches are screwed to a bracket which is mounted using the bolts of the pillow bearings. The limit switches are triggered by hose clamps that are clamped to the 25mm shaft which is rotated by the motor. They are set to trigger when the solar panels are one inch from the post.

![Figure 12: Motor Box Contents](image)

4.1.3 Improvements

Once the mechanical mechanism was built, it was found to have a lot of movement when there should not have been. Looking into the system, there were a few key issues that were addressed. The first of these issues was set screw rounding on the motor shaft seen in Figure 13. The end of the set screw was the only part making contact on the motor shaft, translating the movement to the rest of the system. Since the surface area in contact was so small, the starting and stopping forces exerted to the area was causing the shaft to oval out, creating extra movement that should not be in the system.
To address this issue, the set screw was flattened to create an even contact. Additionally, a tension pin was added through the motor shaft parallel to the set screw, to create more surface area for the stopping and starting forces to push against. These changes can be seen in Figure 14.
There was also a movement within the motor. Tearing the motor apart, it was found that one of the smaller gears had a holding block that was too small. When this slight movement on the smaller gear translated to the large gears, the degrees moved by the shaft was significant. To fix this, a small metal shim was added in the slot to keep the gear from moving left to right. Figure 15 shows this shim under a microscope, as well as the full gear without a microscope.

![Motor Gear Repair](image)

Since the motor purchased was not top quality, there was still some tolerances within the gears after the above discussed issues were fixed. The stacking of the tolerances created about twenty degrees of rotation at the shaft, which in turn reflected the same amount of rotation on the solar panels. The smallest desired movement from our motor was determined to be five degrees, so the limitations of the motor needed to be accounted for. To do this two 36” struts were added to support the panels and provide even pressure on each side of the motor. These struts, seen in Figure 16, have a force of 30 pounds each. The struts provided the precise motor movement desired, but a stronger motor is still recommended. Subsequently, the struts place an upward force on the system, so they must be disconnected from the bracket location on the post before changing the seasonal angle of the system, and then reconnected.
Figure 16: Full System with 36” Struts
4.2 Electrical System

In this section, the electronic system will be discussed. This includes the circuits for motor control, LED control, positional feedback with an accelerometer, switches and other miscellaneous circuit components and systems. The entire schematic for the system can be seen in the appendix.

4.2.1 Motor Control Circuit

The motor control circuit is shown in Figure 17.

![Motor Control Circuit](image.png)

Figure 17: Motor Control Circuit *(C1/C2 included per instruction in [18])

The motor is a 12V worm gear motor (part no. TS-58GZ868-527) in order to hold the solar panel in position even without feeding the motor current. There is an 8A slow-blow fuse attached to the motor. The maximum current draw on the motor is a 6A stall current. We included this so that if the motor is stalled for a long period of time, the fuse will blow because something is clearly wrong, such as it is stuck on something or is driving itself in to the post.

The motor is being controlled by the LMD18201. This motor driver was chosen because it can handle up to 8A and the stall current of the motor is 6A. There are three logic inputs. The motion of the motor can be determined by Table 1.

<table>
<thead>
<tr>
<th>Brake</th>
<th>Direction</th>
<th>Motor Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW</td>
<td>LOW</td>
<td>Forward (clockwise) movement</td>
</tr>
<tr>
<td>LOW</td>
<td>HIGH</td>
<td>Backward (counterclockwise) movement</td>
</tr>
<tr>
<td>HIGH</td>
<td>LOW</td>
<td>STOPPED</td>
</tr>
<tr>
<td>HIGH</td>
<td>HIGH</td>
<td>STOPPED</td>
</tr>
</tbody>
</table>

Table 1: Motor Control Chart

The other logic input is the Pulse Width Modulation pin. The larger the duty cycle of the signal put on this pin, the faster the movement of the motor.
4.2.2 Positional Feedback
In order to complete our control system, feedback was needed in terms of the angle of the solar panel at any given time. Several options were considered, including a potentiometer and a motor encoder. Eventually an accelerometer was chosen, the ADXL335 which is a breakout board from Adafruit™. It is mounted on the back of the solar panel in a weatherproof box. It outputs an analog voltage based on the projection of gravity in that axis. For example, if the board was flat on a table, it would output a maximum value in the z direction and zero in the x and y direction. The schematic for the system is shown in Figure 18.

![Accelerometer Circuit](image)

Figure 18: Accelerometer Circuit

Unfortunately, these analog values fluctuated rapidly, which would not yield a steady or reliable angle computation. In order to solve this problem, a moving average function was implemented to smooth the results. Figure 19 shows the output of such a function.
While the column on the left fluctuates rapidly, the average column on the right stays constant. This above simulation was done by taking the average of the last hundred values. The tradeoff is that with the more values there are, the more accurate it is, but it takes longer to reach a steady state value, the real angle of the solar panel. This means it may not update quickly enough to move the motor without hitting the post. With some empirical testing, using 80 values seemed to be a sweet spot of accuracy and response time.

The analog values cannot be converted to angles by an equation, so they were done empirically. The solar panel was put at the maximum angle for each season based on the Table 2.

<table>
<thead>
<tr>
<th>Season (Angle from vertical)</th>
<th>Minimum Angle</th>
<th>Maximum Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winter (42°)</td>
<td>-28°</td>
<td>28°</td>
</tr>
<tr>
<td>Fall/Spring (52°)</td>
<td>-34°</td>
<td>34°</td>
</tr>
<tr>
<td>Summer (62°)</td>
<td>-37°</td>
<td>37°</td>
</tr>
</tbody>
</table>

Table 2: Maximum Season Angles

At the maximum and minimum angle, the values for the y-direction were recorded. From there, the “map” function of the Arduino® was used. Figure 20 shows an example of how that was done.
Figure 20: Example of Angle Mapping

The above example was printed on the Arduino® Serial Monitor. The minimum and maximum angles for winter (±28º) are shown in columns 4 and 5, while the minimum and maximum y-value readings are shown in columns 2 and 3. The value for the moving average function is shown in the first column and was then mapped between the yMin and yMax to between minAng and maxAng. The output is in column 6 which is 9º. This was used to calculate the angle in all seasons where yMin, yMax, minAng, and maxAng are changed when the x value (not shown) significantly changes to another season. It should also be noted that in the code each season is mapped between 0º and MAX_Season_Ang, where MAX_Season_Ang is equal to two times maxAng.

4.2.3 LED Control Circuit

Two strands of RGB (Red Green Blue) LEDs were used in this project so that the sculpture can light up two different colors. The schematic for the control circuit is shown in Figure 21.
Each strand has four wires. Since these are common cathode LEDs, the common pin goes to 12V from the battery. Then each other pin helps control the amount of red, green, and blue in the emitted light. These pins go through a transistor to ground. When the base of the transistor is on, the LED current is given a path to ground. It should be noted that there is no current limiting resistor at the emitter of the transistor. This was originally overlooked, and testing was done to show that the output of the buffer (SN74HCT541N) was not pulling the maximum current it is capable of. It is rated at 35mA and it was only outputting 21mA. This is indicative of the fact that it is not pulling a maximum current, so that the transistors will not overheat and fail. It is assumed that there are resistors in line with the diodes inside the LED strips which limit the current naturally. In order to be safe, current limiting resistors are a suggested recommendation for any project improvements.

In order to change the amount of each color, a PWM signal is sent to the base of each transistor. The lower the duty cycle, the more often the LED is given a path to ground and therefore the greater intensity of that color that is in the final emitted color.

Table 3 was used to show the colors Red, White, Blue, Coloma Green, and Coloma Gold. The maximum value for any color is 255 because it is one byte of data.
It can be seen that in order to get red, a maximum value is put on red and none on blue and green. Using this same principle, the values for green and gold can be found. A picture was taken from Coloma High School’s website and put into a photo editing software. From that software, the hex values for the school colors were found and then used in the software for controlling the lights.

4.2.4 Connections
Connections with the LEDs were often intermittent and failed occasionally. The LED strips were meant to be cut and then soldered to. When this was done, the connections would fall off and fail. Finally, a solution was arrived at and is shown in Figure 22.

![Figure 22: LED Connections](image)

The wires were poked through a hole in the LED strips, soldered to both sides and then soldered to a prototyping board. Then the strips were secured to this board using screws and washers. This worked well for demonstration purposes, but would be too much of a hassle for Coloma High School for each light strand they install. The recommendation is that Coloma High School buy the LEDs at the required length and do not cut or solder to the strip itself, only the attached wires.

Connectors were used to easily attach and remove the LEDs as well as other parts like the motor, limit switches, and others. Molex connectors were used for this task. There is a male and female plastic connector and metal wire crimps to connect and solder to wires. These can be seen in Figure 23.
4.2.5 Switches

Many switches were utilized in this system, some for safety reasons and some to let users interface with the system.

The first type of switch is a lighting control switch. These are located on the outside of the box to enable users to control the color and function of the lights without opening the enclosure. These switches are shown in Figure 24.

These switches can be seen in the full system schematic in the appendix connected to the Arduino® pins 32, 34, and 26. The options are shown in Table 4 with a switch down being represented by a 0 and a switch up being represented by a 1.
The functions in the fifth column change how the colors are displayed. “Solid” keeps the color at a solid color, “Blink” blinks the color on and off, and “Fade” takes the light from a low intensity to a high intensity and back down.

There is also a kill switch in line with the motor which was originally installed for testing purposes. If the motor kill switch is flipped, the motor will not run.

There is also a System Mode Switch which is in the system schematic connected to pins 52 and 53. In Position One, the system is in remote mode which is discussed in the programming section. In Position Two, the system is in Normal Mode, which involves tracking and lighting on a timer. In Position Three, the system is off; the motor and lights are not running. In this mode, the battery will charge, but no excess power will be depleted other than the power needed to run the Arduino®.

Finally, limit switches were installed in the motor box to prevent the system from moving the solar panel out of range and damaging the mechanism. The code that moves the accelerometer should protect for this as well, but these will be a backup. The switches are shown in Figure 25.
When the lever on the top left is depressed, the bottom two terminals are connected and the switch outputs a high signal. This causes an external interrupt which stops motion and enters a “Recovery Mode” which is discussed in the programming section.

4.2.6 Miscellaneous Circuit Components
For the tracking algorithm, the solar panel voltage is needed. In order to do this, the approximately 20V solar panel voltage goes through a voltage divider to scale it down to be between 0V and 3V. The schematic is shown in Figure 26.

![Figure 26: Solar Panel Input](image)

In order to determine the voltage in the code, first the analog value was turned into a voltage by multiplying by a factor of $3.3V/1023$. Then it was multiplied by $3.17M/470k$ to compute the solar panel voltage and that value was used in the computations.

Two voltage regulators were used in this system. One 9V regulator was used to power the Arduino® through a DC power jack. A 3.3V regulator was used to power the buffers, photovoltaic isolators, and logic signals. Each regulator had a 1000µF electrolytic capacitor across the Vcc to ground. At each 3.3V input to an IC, there was a 0.1µF ceramic capacitor. These capacitors are used to eliminate ripple and noise on the DC voltage output.

A Wanderer® charge controller came with the solar panel. It has inputs for the positive and negative terminals of the solar panels as well as terminals for the battery. It also has a terminal for a temperature sensor, which was not used in this project. The manual indicates that the charge controller has overcharge and short circuit protection. Another recommendation for this project could be to investigate this charge controller and whether or not it can be replaced or improved.

4.2.7 Electronic Build
The circuit components were soldered to a prototyping board, as shown in Figure 27.
The inside of the enclosure is shown in Figure 28.

Figure 27: Prototyping Board

Figure 28: Finished Ground Plate
4.3 Programming

An Arduino® Due was interfaced with the RGB LEDs and worm gear motor to regulate RGB color control and solar panel tracking. The program running on the Arduino® Due needed to handle monitoring various sensor voltages, rotating the solar panel each hour, and controlling the RGB LEDs. External and timer interrupts were implemented in order to control all of these functions. Along these functions, a recovery and a remote mode were created to make a more robust program. Appendix 9 features 3 tables covering the functions, pins, and variables used in the C Code.

4.3.1 Timer Interrupts

Timer Interrupts use the internal clock of a microcontroller to periodically complete a task. The benefit of using the timer interrupt is that it allows the microcontroller to do other tasks while the timer is running. In order to run timer based interrupts, the DueTimer.h library needs to be included. The Arduino® has a maximum timer interrupt period of 5 seconds. The HourISR function is designed to prolong this 5 second value to one hour by incrementing an unsigned integer called DTANGLE by 1 every 5 seconds. When the DTANGLE value reaches 720 an hour has passed. Along with the HourISR, the solar panel output voltage and the solar panel angular position are monitored with timer based interrupts every 20ms the current voltage value is updated in both the COMPVOLT and ANGLE functions.

4.3.2 External Interrupts

External interrupts are triggered by a change in a digital GPIO input reading. External interrupts also allow the Arduino® to complete other tasks while the state of the GPIO remains the same. The lighting controller toggle switches, remote switch, turn off switch, and both limit switches are all configured as external interrupts. When the value of one of these switches changes the Arduino® will enter the interrupt service routine specific to the switch toggled. For example, the TurnOff switch inside of the enclosure allows for the user to halt all the solar tracking during routine maintenance. When the TurnOff switch is set HIGH, the program enters the TURNOFF() interrupt service routine. The TURNOFF function stays in a while loop until the TurnOff switch is set back to LOW.

4.3.3 Analog Inputs

The Arduino® reads the voltage from the solar panel and the accelerometer. The COMPVOLT function implements the Arduino® ’s analog to digital converter to quantize the solar panel output voltage. This voltage is then multiplied by .02215 to scale the voltage to a value between 0 and 22V. The ANGLE function also reads in three analog voltages representing the amount of gravity experienced in the x, y, and z directions. The x, y, and z component voltages are stabilized using a moving average filter. The moving average filter stores the last 100 voltage readings in an array and calculates the average. This average is then mapped to a value between -90° and +90° In order to account for the three seasonal angles, the algorithm uses the value in the x direction. The mapping values change in the Y direction depending on the magnitude of the x component. This results in a more robust calibration for the solar panel in the y direction. The solar panel The solar panel voltage and the solar panel angle are stored as global variables.
4.3.4 Tracking Algorithm
The Arduino® controls the solar panel tracking during the day. The Tracker function begins when the counter in the HourISR function reaches 720, signifying that an hour has passed. When an hour has passed the Arduino® will rotate the solar panel 15° to the west. The program is designed to move the solar panel 15° to remain perpendicular to the sun which moves approximately 15° per hour. After moving, the Arduino® stores the voltage of the solar panel and then rotates the solar panel 5 more degrees. If the voltage from the solar panel increases, the Arduino® will continue to rotate in 5 degree increments. If the voltage from the solar panel decreases, the Arduino® will quit moving the solar panel and start waiting another hour to track again. Figure 29 is a flow chart for the solar tracking algorithm.

![Solar Tracking Algorithm Logic](image)

4.3.5 Sunrise and Sunset Detection
The Arduino® is also programmed to detect sunrise and sunset. Sunset is detected when the accelerometer has reached its maximum angle of rotation and when the voltage from the solar panel is less than 13 Volts. After Sunset is detected, the solar panel stays in the same position for 3 hours so that it can capture more energy from the sun while it sets. After 3 hours passes, the solar panel returns to its starting position and waits for sunrise. Sunrise is detected when the solar panel voltage exceeds 17 Volts. When the voltage reaches 17V the Arduino® returns to tracking the sun. Figure 30 is a flow chart for the logic behind sunrise and sunset detection.
4.3.6 Light Controller
The Arduino® controls the RGB LEDs lighting the sculpture. The color of the RGB LED depends on the duty cycle set by the Arduino®. There are 3 different color combinations and three different modes available. The mode and color combination depends on the state of three external toggle switches. A 3-bit value is generated by shifting the second and third toggle switch values to the left. 3-bits generate 8 possible combinations. The color mode functions are SOLID, FADE, and BLINKY, these functions read in 6 inputs representing the red, green, and blue duty cycles for two LED strands. The SOLID function outputs a constant color. The FADE function divides the duty cycle of every color by an incrementing value. Dividing the duty cycle creates a dimming effect. As the value of the counter increases and decreases the lights appear to fade in and fade out. The BLINKY function blinks the LEDs on and off. Each time the LEDs blink the colors of strand one and strand two switch. The three external switches are assigned as external interrupts instantly updating the 3-bit value when toggled.

4.3.7 Recovery Mode
Limit switches were used to prevent the solar panel from moving outside of its designed range. The limit switches are assigned as external interrupts. The Arduino® is programmed to autonomously recover when one of the limit switches is pressed. When one of the limit switches is toggled, the Arduino® automatically breaks out of the function that is moving the motor. After stopping the motor, Arduino® then moves the solar panel in the opposite direction for 100ms. Every time the limit switch is pressed the amount of recoveries is updated. Figure 31 shows the logic behind the recovery function.
4.3.8 Remote Mode

A software remote was created to control different processes handled by the Arduino®. This software was primarily made for testing the motor, but the versatile nature of the software controller allowed for the addition of a light controller and a diagnostics display. The remote communicates over the serial monitor of the Arduino®. Commands are sent using a laptop keyboard. The motor controller receives a position and speed from the keyboard and then moves the solar panel to that position at the desired speed. An RGB LED controller was also programmed. When in the RGB LED mode, the user uses capital and lowercase “r”, “g”, and “b” keys to increase and decrease the duty cycles respectively. The remote mode also provides the user with system diagnostics. When the diagnostics mode is entered, the Arduino® will display the solar panel voltage, the current value of the HourISR counter, the solar panel position, and the amount of recoveries. This allows the user to verify that the system is functioning properly. Figure 32 depicts the interface that the user sees on their laptop screen.

Figure 32: Visual Interface of Remote Mode.
5.0 VALIDATION

In this section, a review of the original specifications is done. Then testing is discussed, including the primary function testing as well as system level testing.

5.1 Specifications Met

<table>
<thead>
<tr>
<th>Specification #</th>
<th>Category: Subject</th>
<th>Original Spec. Met</th>
<th>Reason for Change (if present)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1.1</td>
<td>Size: Lighting</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.1.1.2</td>
<td>Size: Medallions</td>
<td>Altered</td>
<td>School will add medallions at a later date.</td>
</tr>
<tr>
<td>3.1.2.1</td>
<td>Electrical: Solar Output</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.1.2.2</td>
<td>Electrical: Battery Output</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.1.2.3</td>
<td>Electrical: Lighting</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Environmental: External</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Environmental: Electrical</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.1.4.1</td>
<td>Location: Power Source</td>
<td>Altered</td>
<td>School will determine final install locations.</td>
</tr>
<tr>
<td>3.2.1.1</td>
<td>Solar Panel: Track</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.2.1.2</td>
<td>Solar Panel: Charge</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.2.2.1</td>
<td>Lighting System: Medallions</td>
<td>No</td>
<td>No Change</td>
</tr>
<tr>
<td>3.2.2.2</td>
<td>Lighting System: School</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.2.2.3</td>
<td>Lighting System: Other</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.2.2.4</td>
<td>Lighting System: Replacing</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.2.3.1</td>
<td>Control Box: Motor Control</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.2.3.2</td>
<td>Control Box: Convert Solar</td>
<td>Yes</td>
<td>No Change</td>
</tr>
<tr>
<td>3.2.3.3</td>
<td>Control Box: Convert Battery</td>
<td>Yes</td>
<td>No Change</td>
</tr>
</tbody>
</table>

Table 5: Specifications Comparison Table
5.2 Testing

5.2.1 Primary Function Testing
The program running on the Arduino®™ was tested in 3 different segments before testing the entire system. The program tracking function, night function, and lighting control were tested individually. These tests reference information from the Arduino® serial monitor, this information can be viewed in appendices 4-6. The test code generated for each of the three tests is also available in these appendices.

5.2.1.1 Tracking Function
The tracking function is responsible for finding the optimal solar panel position every hour. In order to test, the one-hour interval was scaled down to 50 seconds. The accelerometer was simulated using a potentiometer. A DC voltage source was used to simulate the voltage value of the comparator circuit. The while loop in the function will be tested under two cases:

1. The program detects a voltage drop in the comparator circuit.
2. The program detects a constant voltage/ increasing voltage from the comparator circuit.

The comparator circuit and the accelerometer were both simulated using a potentiometer. The SN754410NE H Bridge was used for motor control. All components were buffered using the 74HCT541D noninverting buffer. Figure 33 shows the schematic of the system that was assembled.

![Figure 33: Tracking Function Validation Circuit](image)

Figures 34 show the actual circuit setup as well as the DC voltage sources used to simulate the comparator circuit and accelerometer.
5.2.1.1.1 Serial Monitor Part 1

In test one the microcontroller detects a lower voltage from the comparator circuit and exits the while loop after one iteration. The following information was pulled from the Arduino® IDE serial monitor. Using “Serial.println()” one can print values as well as text onto the monitor. The points highlighted in the tracking function are: the beginning, the first 10° angle change, entering the while loop, fine tuning the solar panel position, and the end of the program.

5.2.1.1.2 Serial Monitor Part 2

In this test the microcontroller never detected a reduction in the solar panel output voltage, causing it to precisely detect the optimal angle 4 times. One can see from the serial monitor that
the FourTries counter incremented all the way to three before exiting the while loop and ending the program.

5.2.1.1.3 Tracking Function Test Conclusion

The tracking function performed properly in both tests. During the testing the 74HCT541D inverter interfered with the Arduino®’s voltage reading. The test also allowed for studying how the Arduino® quantizes the analog voltage inputs. The Arduino® assigns a value of 1023 for a 5 Volt input. While testing the final program the values used to fine tune the solar panel angle will have to be scaled to the actual accelerometer signal value.

5.2.1.2 Night Function

Test 2 validates the performance of the night function. This function is responsible for detecting both the sunrise and the sunset. In order to test the function, the three-hour counter used during sunset will be scaled down to 1 minute. While the program is in sunset or sunrise mode a simple version of the lighting code will be enabled. During the test the following events should occur:

1. Set the accelerometer voltage to 2.5 volts with the DC voltage source.
2. Set the comparator voltage to 1.6 Volts.
3. The program enters night function.
4. The NIGHTHOURS counter is set to 0.
5. The LED turns on for one minute.
6. The motor rotates until the tilt angle voltage is set to 0.
7. The LED will turn on.
8. The function ends when the comparator voltage is set above 1.6 Volts.

The Arduino® will be interfaced with the SN754410NE H Bridge, the 74HCT541D noninverting buffer, an LED, a 1k resistor and three DC variable voltage sources. The circuit schematic below shows the circuit configuration for this test. Figure 35 shows the schematic used for validation.
Figure 35: Validation of Night Mode Circuit

Figure 36 shows the circuit setup along with the DC voltage sources used to simulate the solar panel and accelerometer voltages.
The night function performed properly. During the test, the program was able to identify sunset and sunrise based on the accelerometer voltage and comparator circuit voltage. For the final code, all GPIO pins must be set low after sunrise is detected ensuring that they are not left on when sunrise is detected.

5.2.1.3 Lighting Control Using External Interrupts

The RGB LED’s have multiple color combinations and settings that can be selected by the user via 3 external toggle switches. These switches trigger an external interrupt in the Arduino®
instantly changing the lighting configuration based on user selection. 3 switches generate 8 possible combinations. In order to validate that the interrupts are working and the SVAL program is functioning properly the following test will be conducted:

1. Bread board three switches to the Arduino® in the circuit configuration shown in Figure 37.
2. Toggle the three switches making every combination 0-7.
3. Print the value of the lighting configuration using the Serial Monitor.

The external interrupts functioned properly. The Arduino® accurately generated a 3-bit number based on the input of the toggle switches.
5.2.2 System Testing

5.2.2.1 Programs Implemented in System Level Testing

Two programs were developed to conduct testing on the solar panel tracking system. The first program was a data logger. The data logger program implemented timer based interrupts to record the voltage of the solar panel every 30 seconds. This voltage was then printed in the serial monitor. The second program was a motor script. The motor script moved the solar panel from start position to end position repetitively while incrementing the duty cycle each test. The C code for these programs are available in appendices 7 and 8 respectively.

5.2.2.1 System Level Testing

On February 19th, 2016 tests were done at Coloma High School to monitor the voltage of the solar panel as the sun rose. Figure 38 shows the results of that test.

![Figure 38: Sunrise Voltage Testing, Coloma High School](time_of_day_graph)

While these tests were done with different solar panels, the same characteristic curve is expected on the used solar panel. Sunrise on that day was at 7:36am and it can be seen after that a plateau is seen of the voltage. This test was used to help determine the threshold voltage that would cause the solar panel tracking system to begin each day.

On April 17th, 2016, testing was done at sunset at the Parkview Engineering Campus. This test was done using the Data Logger function discussed above. The results of this test are shown in Figure 39.
On this day, sunset was at 8:27pm. Several minutes after that, a characteristic decline in solar panel voltage as seen. This helped to determine the threshold voltage which would cause the system to go into Night Mode.

Testing was also done on April 18th during the middle of the day. This was sunny with no clouds in the sky. The results of this testing is shown in Figure 40.
This testing shows a fairly periodic dip in solar panel voltage. While there is no evidence as of what caused this, some hypotheses can be made. One possible cause is the way in which the solar panel charges the battery through the charge controller. It is possible that there is a voltage being formed in the charge controller which is causing these unexpected readings. Another possible reason is that there were clouds in the sky which registered in the solar panel voltage, but could not be viewed in the sky. Finally, since these data points were taken using the Data Logger function, it is possible there is some sort of overflow in the function which causes these periodic dips. One recommendation for future work is to find the cause of this abnormality.
This project began as a need from Coloma High School to light a sculpture with their school colors. Solar panel tracking was implemented to optimize the solar panel power generation. The design allowed for solar panel rotation and tracking, as well as lighting controls for users. The Remote Mode allowed for a variety of user interactions based on the Serial Monitor input from the Arduino® Due microcontroller.

Overall, the specifications for this project were met and the project provides a system for Coloma High School to implement. Also, this project is a solid basis upon which to base new senior design projects, such as optimizing the solar tracking, building a new charge controller, and turning the prototyping board into a printed circuit board.
8.0 RECOMMENDATIONS

As the solar tracking and RGB LED control project continues, the system could be improved in three ways. The first improvement would be using a more robust motor. The current motor has a lot of slop in the gears, a higher quality motor would allow for a more accurate solar tracking system. The second improvement could be made by manufacturing a printed circuit board instead of making the circuit by hand on a prototyping board. Lastly the system could be improved through optimizing the solar tracking algorithm. One could conduct tests to calculate the optimal amount of times to track the sun per day. Through these recommendations one could thoroughly improve the current solar tracking and RGB LED control system.
9.0 ACKNOWLEDGEMENTS

Our team would like to thank all of those who have helped support this project. Coloma High School has presented us with this opportunity and provided the financial backing necessary. Professors Gejji and Miller have given their knowledge and insight to ensure that success of this project. Jonathan Rhodes provided the mechanical knowledge and support throughout the development of the mechanical mechanism. Louis Rhodes and his team at General Motors took our model and parts, turning our idea into a working mechanical system. In addition, their donation of solar panels greatly increased the capabilities of our system. The work of these contributors was greatly appreciated.
References


Appendices

Appendix 1: Circuit Schematic
## Appendix 2: Bill of Materials

<table>
<thead>
<tr>
<th>Description</th>
<th>Identifier in Schematic</th>
<th>Part No</th>
<th>Used in System</th>
<th>Number</th>
<th>Cost per part</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outdoor RGB LED Strips</td>
<td>LED1, LED2</td>
<td>WFLS-RGB30-BK</td>
<td>Full System</td>
<td>2</td>
<td>$104.95</td>
<td>$209.90</td>
</tr>
<tr>
<td>12 V 60AH Deep Cycle AGM Solar Battery</td>
<td>L9</td>
<td>APX12-75</td>
<td>Full System</td>
<td>1</td>
<td>$123.99</td>
<td>$123.99</td>
</tr>
<tr>
<td>Arduino Due</td>
<td>AT91SAM3X8E</td>
<td>Electronic System</td>
<td>4</td>
<td>$6.99</td>
<td>$27.96</td>
<td></td>
</tr>
<tr>
<td>Grand General 16 Gauge Wire</td>
<td>55231</td>
<td>Electronic System</td>
<td>4</td>
<td>$6.99</td>
<td>$27.96</td>
<td></td>
</tr>
<tr>
<td>20x06xl2 NEMA 4/4 Enclosure</td>
<td>SEAL-20EL1612LP</td>
<td>Enclosure</td>
<td>1</td>
<td>$211.00</td>
<td>$211.00</td>
<td></td>
</tr>
<tr>
<td>17x13 Subpanel</td>
<td>SEAL-20P1666</td>
<td>Enclosure</td>
<td>1</td>
<td>$27.00</td>
<td>$27.00</td>
<td></td>
</tr>
<tr>
<td>Trim Shell Kit for Type 4 Enclosure Models</td>
<td>SEAL-DS1654</td>
<td>Enclosure</td>
<td>1</td>
<td>$37.00</td>
<td>$37.00</td>
<td></td>
</tr>
<tr>
<td>12V 60AH Deep Cycle AGM Solar Battery</td>
<td>U9</td>
<td>APX12-75</td>
<td>Full System</td>
<td>1</td>
<td>$123.99</td>
<td>$123.99</td>
</tr>
<tr>
<td>12V 60AH Deep Cycle AGM Solar Battery</td>
<td>U9</td>
<td>APX12-75</td>
<td>Full System</td>
<td>1</td>
<td>$123.99</td>
<td>$123.99</td>
</tr>
<tr>
<td>Arduino Due</td>
<td>AT91SAM3X8E</td>
<td>Electronic System</td>
<td>4</td>
<td>$6.99</td>
<td>$27.96</td>
<td></td>
</tr>
<tr>
<td>Grand General 16 Gauge Wire</td>
<td>55231</td>
<td>Electronic System</td>
<td>4</td>
<td>$6.99</td>
<td>$27.96</td>
<td></td>
</tr>
<tr>
<td>20x06xl2 NEMA 4/4 Enclosure</td>
<td>SEAL-20EL1612LP</td>
<td>Enclosure</td>
<td>1</td>
<td>$211.00</td>
<td>$211.00</td>
<td></td>
</tr>
<tr>
<td>17x13 Subpanel</td>
<td>SEAL-20P1666</td>
<td>Enclosure</td>
<td>1</td>
<td>$27.00</td>
<td>$27.00</td>
<td></td>
</tr>
<tr>
<td>Trim Shell Kit for Type 4 Enclosure Models</td>
<td>SEAL-DS1654</td>
<td>Enclosure</td>
<td>1</td>
<td>$37.00</td>
<td>$37.00</td>
<td></td>
</tr>
<tr>
<td>12V 60AH Deep Cycle AGM Solar Battery</td>
<td>U9</td>
<td>APX12-75</td>
<td>Full System</td>
<td>1</td>
<td>$123.99</td>
<td>$123.99</td>
</tr>
<tr>
<td>12V 60AH Deep Cycle AGM Solar Battery</td>
<td>U9</td>
<td>APX12-75</td>
<td>Full System</td>
<td>1</td>
<td>$123.99</td>
<td>$123.99</td>
</tr>
<tr>
<td>Arduino Due</td>
<td>AT91SAM3X8E</td>
<td>Electronic System</td>
<td>4</td>
<td>$6.99</td>
<td>$27.96</td>
<td></td>
</tr>
<tr>
<td>Grand General 16 Gauge Wire</td>
<td>55231</td>
<td>Electronic System</td>
<td>4</td>
<td>$6.99</td>
<td>$27.96</td>
<td></td>
</tr>
<tr>
<td>20x06xl2 NEMA 4/4 Enclosure</td>
<td>SEAL-20EL1612LP</td>
<td>Enclosure</td>
<td>1</td>
<td>$211.00</td>
<td>$211.00</td>
<td></td>
</tr>
<tr>
<td>17x13 Subpanel</td>
<td>SEAL-20P1666</td>
<td>Enclosure</td>
<td>1</td>
<td>$27.00</td>
<td>$27.00</td>
<td></td>
</tr>
</tbody>
</table>

**Total** $1,505.79
Appendix 3: Solar Tracking Algorithm and RGB LED Controller Program
#include <DueTimer.h>

/* *****Version 5.0**********
* Solar Tracking Algorithm and RGB Light Controller
* Middleton
*
* 5.0
* 4.15.16
* Comparator voltage and Accelerometer sensors are now checked via timer interrupt
* Recovery information is displayed in the diagnostics
* Seasonal angle logic incorporated and volatile variable MAX_Season_Angle used.
* Amount of iterations in FourTries reduced to 2.
*
* 4.0
* 4.13.16
* Fully functional ready for System Level testing
* moving average filter implemented to stabilize accelerometer
* and comparator voltage readings.
* Recovery mode improved to move the system out of recovery mode autonomously.
* Lightcontroller always on.
*
* 3.2.1
* 4.12.16
* Recovery mode and limit switches integrated into algo.
*
* 3.2
* 4.12.16
* Timer1 interrupt library changed to DueTimer. Compatible with Arduino® DUE.
* BLINKY() function added as third lighting mode
* FADE algo improved.
* PWM pin renamed SpeedSelect due to library definition conflicts
* Angle program integrated into algo.
* COMPVOLT and ANGLE functions improved to interrupt service routines updated every 10 ms
* Diagnostics added to Middleton_CONTRL
*
* 4.9.16
* Software remote controller improved to allow for RGB Led color change.
* FADE and SOLID light functions were updated to accommodate for two LED strips.
*
* 4.7.16
* Software Remote Controller added with ability to move solar panel to desired position
* Intensity Pins removed due to circuit design
* Lighting control fade function modified.
* Functions added "CleanUp" "KeyInputHandler" and "MIDDLETON_CNTRL"
* V2.2 3.7.16
* Solar Tracking Algorithm and RGB Light Controller
* Middleton
* created 3.6.16
* Pin numbers assigned
*
* V2.1 3.1.16
* Diagnostics removed
* Day counter removed
* Full lighting functionality added
* Time based motor control added to accommodate for accelerometer latency.
*
*/

// functions used
void HourIsr();
void SVAL();
void LIGHTCONTROLLER();
void Tracker();
void movemotor(uint8_t NEWANGLE);
void COMPVOLT();
void FADE(uint8_t x, uint8_t y, uint8_t z, uint8_t x2, uint8_t y2, uint8_t z2);
void SOLID(uint8_t x, uint8_t y, uint8_t z, uint8_t x2, uint8_t y2, uint8_t z2);
void BLINKY(uint8_t x, uint8_t y, uint8_t z, uint8_t x2, uint8_t y2, uint8_t z2);
void ANGLE();
void CleanUp();
uint8_t KeyInputHandler();
void Middleton_CNTRL();
void TURNOFF();
void UpdateColor();
void Recovery();
float Middleton_Moving_Average(uint16_t x, uint8_t opto);

//Global Variables
volatile uint16_t DTCOUNTER;
volatile uint8_t DTANGLE;
volatile float yRead;
volatile uint16_t NIGHTOURH;
volatile uint8_t LIGHTCONFIG;
volatile uint8_t xAng;
volatile uint8_t yAng;
volatile uint8_t zAng;
volatile float SCALEDVOLT;
volatile uint8_t LimiThrow;
volatile uint8_t MAX_Season_Angle;
// Assign output Pin names
uint8_t RED1 = 12;
uint8_t GREEN1 = 11;
uint8_t BLUE1 = 10;
uint8_t RED2 = 8;
uint8_t GREEN2 = 7;
uint8_t BLUE2 = 6;
uint8_t Dir = 5;
uint8_t Brake = 4;
uint8_t SpeedSelect = 3;

// Assign Input Pin names
char Comparator = A0;
uint8_t TOGGLE1 = 32;
uint8_t TOGGLE2 = 34;
uint8_t TOGGLE3 = 36;
uint8_t KILLSWITCH1 = 46;
uint8_t KILLSWITCH2 = 47;
uint8_t AllOff = 53;
uint8_t Controller = 52;
char xPin = A1;
char yPin = A2;
char zPin = A3;

// Remote Vars
volatile uint8_t Speed;
char SpeedH;
char SpeedT;
char SpeedO;
uint8_t ANG;
volatile char ANGH;
volatile char ANGT;
volatile char ANGO;
uint8_t OPTION;
uint8_t state;
uint8_t wtt;
uint8_t CIn;
volatile uint8_t t DutyR = 0;
volatile uint8_t t DutyB = 0;
volatile uint8_t t DutyG = 0;
uint8_t t Mode;

// Array initializations
uint16_t MMAx[100];
uint16_t MMAy[100];
uint16_t MMAz[1000];
void setup()
{
    // Outputs
    pinMode(RED1, OUTPUT);
    pinMode(GREEN1, OUTPUT);
    pinMode(BLUE1, OUTPUT);
    pinMode(RED2, OUTPUT);
    pinMode(GREEN2, OUTPUT);
    pinMode(BLUE2, OUTPUT);
    pinMode(Dir, OUTPUT);
    pinMode(Brake, OUTPUT);
    pinMode(SpeedSelect, OUTPUT);
    // Inputs
    pinMode(TOGGLE1, INPUT);
    pinMode(TOGGLE2, INPUT);
    pinMode(TOGGLE3, INPUT);
    pinMode(Controller, INPUT);
    pinMode(AllOff, INPUT);
    pinMode(KILLSWITCH1, INPUT);
    pinMode(KILLSWITCH2, INPUT);
    pinMode(Comparator, INPUT);
    pinMode(xPin, INPUT);
    pinMode(yPin, INPUT);
    pinMode(zPin, INPUT);

    // Interrupt initializations
    Timer1.attachInterrupt(HourIsr); // tell the timer to go to ISR
    Timer1.start(5000000); // set timer to 5 seconds
    Timer2.attachInterrupt(ANGLE);
    Timer2.attachInterrupt(COMPVOLT);
    Timer2.start(20000); // check every 10 ms
    attachInterrupt(digitalPinToInterrupt(TOGGLE1), SVAL, CHANGE); // Change the Switchval variable on change
    attachInterrupt(digitalPinToInterrupt(TOGGLE2), SVAL, CHANGE); // Change the Switchval variable on change
    attachInterrupt(digitalPinToInterrupt(TOGGLE3), SVAL, CHANGE); // update switchval variable on change
    attachInterrupt(digitalPinToInterrupt(KILLSWITCH1), RECOVERY, HIGH);
    attachInterrupt(digitalPinToInterrupt(KILLSWITCH2), RECOVERY, HIGH);
    attachInterrupt(digitalPinToInterrupt(AllOff), TURNOFF, HIGH);
    attachInterrupt(digitalPinToInterrupt(Controller), Middleton_CNTRL, HIGH);
    Serial.begin(9600);
}

void loop()
{
void RECOVERY()
{
  if(digitalRead(KILLSWITCH1) | digitalRead(KILLSWITCH2) == 1)
  {
    LimitThrow++;
    digitalWrite(Brake, HIGH);
    if(digitalRead(KILLSWITCH1) == 1)
    {
      digitalWrite(SpeedSelect, Speed);
      digitalWrite(Dir, LOW);
      digitalWrite(Brake, LOW);
      delay(50);
      digitalWrite(Brake, HIGH);
    }
    if(digitalRead(KILLSWITCH2) == 1)
    {
      digitalWrite(SpeedSelect, Speed);
      digitalWrite(Dir, HIGH);
      digitalWrite(Brake, LOW);
      delay(50);
      digitalWrite(Brake, HIGH);
    }
  }
  else
  {
  }
}

void TURNOFF() // All functions halted until switched back on.
{
  while(digitalRead(AllOff) == 1)
  {
    // Do nothing when AllOff is high.
    DTCOUNTER = 0;
    NIGHTHOUR = 0;
  }
}

void HourIsr()
{
  DTCOUNTER = DTCOUNTER + 1;
  NIGHTHOUR = NIGHTHOUR + 1;
  if (DTCOUNTER >= 720)
  {
Tracker();
DTCOUNTER=0; //reset the counter back to zero;
}
if ((DTANGLE>=MAX_Season_Angle) && (SCALEDVOLT<13))
{
    NIGHT();
}
}
void Tracker()
{
    uint8_t NewAngle; //Create a variable for the desired panel angle
    uint8_t PVOLT=0; // Create a variable to store the current pannel voltage in
    uint8_t FINETUNEANGLE;
    uint8_t FourTries = 0;

    NewAngle=DTANGLE+10; //create a value for the desired position
    if(NewAngle > MAX_Season_Angle) //protect for positions greater than 150.
    {
        NewAngle = MAX_Season_Angle;
        FourTries = 2;
    }
    movemotor(NewAngle);
    delay(100); //wait 5 seconds to get a panel voltage reading may change depending on ability to detect change
    FINETUNEANGLE = DTANGLE; //set the finetuneangle to current position, prepare to calibrate

    while((SCALEDVOLT>=PVOLT) && (FourTries<2))
    {
        PVOLT = SCALEDVOLT;
        FINETUNEANGLE = FINETUNEANGLE + 5; // increment posiition 5 degrees
        if(FINETUNEANGLE> 150)
        {
            FINETUNEANGLE = 150;
            FourTries = 2;
        }
        movemotor(FINETUNEANGLE);
        delay(100);
        FourTries = FourTries+1; // we will only try and find optimal position 4 times before returning to main
    }
}
void NIGHT()
{
    NIGHTHOUR=0;
while(NIGHTHOUR<2160)
{
    LIGHTCONTROLLER();
    DT COUNTER = 0;
}
movemotor(0);
while(SCALEDVOLT < 17)
{
    LIGHTCONTROLLER();
}
}
void movemotor(uint8_t NEWANGLE)
{
    if(abs(yRead - NEWANGLE) > 3)
    {
        while(yRead < NEWANGLE)
        {
            if(digitalRead(KILLSWITCH1) | digitalRead(KILLSWITCH2) == 1)
            {
                break;
            }
            else
            {
                digitalWrite(Dir, HIGH); // set direction
                analogWrite(SpeedSelect, Speed); // set speed
                digitalWrite(Brake, LOW); // release brake
            }
        }
        while(yRead > NEWANGLE)
        {
            if(digitalRead(KILLSWITCH1) | digitalRead(KILLSWITCH2) == 1)
            {
                break;
            }
            else
            {
                digitalWrite(Dir, LOW); // set direction
                analogWrite(SpeedSelect, Speed); // set speed
                digitalWrite(Brake, LOW); // release brake
            }
        }
        analogWrite(SpeedSelect, 0); // reduce the speed to 0;
        digitalWrite(Brake, HIGH); // Apply the brake;
    }
}
else
{
void COMPVOLT()
{
  float VREADING = analogRead(Comparator);
  SCALEDVOLT = VREADING*3.3/1023*(3.147/.47)*(13.46/12.7);  //a value between 0 and 22
  SCALEDVOLT = Middleton_Moving_Average(SCALEDVOLT, 3);
}

void ANGLE()
{
  xRead = analogRead(xPin);
  yRead = analogRead(yPin);
  zRead = analogRead(zPin);

  xAvg = Middleton_Moving_Average(xRead, 0);
  yAvg = Middleton_Moving_Average(yRead, 1);
  zAvg = Middleton_Moving_Average(zRead, 2);

  if(xAvg >= 580 || xAvg <= 590) //winter (62 degrees)
  {
    yMin = 470;
    yMax = 541;
    MAX_Season_Angle = 56;
  }
  else if(xAvg >= 567 || xAvg < 580) //spring and fall (52 degrees)
  {
    yMin = 454;
    yMax = 556;
    MAX_Season_Angle = 74;
  }
  else if(xAvg >= 555 || xAvg <= 567) //summer (42 degrees)
  {
    yMin = 470;
    yMax = 541;
    MAX_Season_Angle = 68;
  }
  uint8_t minValx = 299;
  uint8_t maxValx = 450;
  uint8_t minValy = 296;
  uint8_t maxValy = 447;
  uint8_t minValz = 303;
  uint8_t maxValz = 453;
  //read the analog values from the accelerometer
  float xRead = Middleton_Moving_Average(analogRead(xPin),0);
```c
float yRead = Middleton_Moving_Average(analogRead(yPin),1);
float zRead = Middleton_Moving_Average(analogRead(xPin),2);
//convert read values to degrees -90 to 90 - Needed for atan2
uint8_t xAng = map(xRead, minValx, maxValx, -90, 90);
yAng = map(yRead, minValy, maxValy, -90, 90);
uint8_t zAng = map(zRead, minValz, maxValz, -90, 90);
DTANGLE = map(yAvg, yMin, yMax, minAng, maxAng);

if(yAng>0)
{
    zAng=180-zAng;
}

void SVAL()
{
    // create a three bit value to represent the 3 toggle switches used for user input.
    LIGHTCONFIG = (digitalRead(TOGGLE1)) | (digitalRead(TOGGLE2)<<1 | (digitalRead(TOGGLE3))<<2; // create a 3 bit lightconfig value
}
void LIGHTCONTROLLER() // this function uses the timer interrupts to turn the lights on 2
hours before sunrise and 2 hours after sunset.
{
    if(LIGHTCONFIG==0)
    {
        SOLID(195,235,5,14,170,8);//SOLID GREEN GOLD
    }
    if(LIGHTCONFIG==1)
    {
        BLINKY(195,235,5,14,170,8); //BLINK GREEN GOLD
    }
    if(LIGHTCONFIG==2) //FADE G G
    {
        FADE(195,235,5,14,170,8);
    }
    if(LIGHTCONFIG==3)
    {
        SOLID(255,0,0,0,0,255); //SOLID RED BLUE
    }
    if(LIGHTCONFIG==4) //BLINK R->B R->W W->B
    {
        BLINKY(255,0,0,0,0,255);
        BLINKY(255,0,0,255,255,255);
        BLINKY(255,255,255,0,0,255);
    }
    if(LIGHTCONFIG==5) //FADE RWB
```
if(LIGHTCONFIG==6)
{
  FADE(254,0,0,254,102,0);
  FADE(254,102,0,254,254,0);
  FADE(254,254,0,0,254,0);
  FADE(0,254,0,0,0,254);
  FADE(0,0,254,150,0,255);
  FADE(150,0,255,238,130,238);
  FADE(238,130,238,255,255,255);
}
if(LIGHTCONFIG==7)
{
  BLINKY(254,0,0,254,102,0);
  BLINKY(254,102,0,254,254,0);
  BLINKY(254,254,0,0,254,0);
  BLINKY(0,254,0,0,0,254);
  BLINKY(0,0,254,150,0,255);
  BLINKY(150,0,255,238,130,238);
  BLINKY(238,130,238,255,255,255);
}
void FADE(uint8_t x, uint8_t y, uint8_t z, uint8_t x2, uint8_t y2, uint8_t z2)
{
    uint8_t state = 1;
    uint8_t count = 1;
    uint8_t laggy = 90;
    uint16_t hold = 900;
    if(state == 1)
    {
      analogWrite(RED1, x2);
      analogWrite(GREEN1, y2);
      analogWrite(BLUE1, z2);
      analogWrite(RED2, x);
      analogWrite(GREEN2, y);
      analogWrite(BLUE2, z);
      delay(hold);
      while(count<25)
      {
        analogWrite(RED1, x2/count);
        analogWrite(GREEN1, y2/count);
        analogWrite(BLUE1, z2/count);
      
```
analogWrite(RED2, x/count);
analogWrite(GREEN2, y/count);
analogWrite(BLUE2, z/count);
count++;
delay(laggy);
}
while(count>1)
{
analogWrite(RED1, x/count);
analogWrite(GREEN1, y/count);
analogWrite(BLUE1, z/count);
analogWrite(RED2, x2/count);
analogWrite(GREEN2, y2/count);
analogWrite(BLUE2, z2/count);
count--;
delay(laggy);
}
analogWrite(RED1, x);
analogWrite(GREEN1, y);
analogWrite(BLUE1, z);
analogWrite(RED2, x2);
analogWrite(GREEN2, y2);
analogWrite(BLUE2, z2);
delay(hold);
state = 0;
}
if(state == 0)
{
while(count<25)
{
analogWrite(RED1, x/count);
analogWrite(GREEN1, y/count);
analogWrite(BLUE1, z/count);
analogWrite(RED2, x2/count);
analogWrite(GREEN2, y2/count);
analogWrite(BLUE2, z2/count);
count++;
delay(laggy);
}
while(count>1)
{
analogWrite(RED1, x2/count);
analogWrite(GREEN1, y2/count);
analogWrite(BLUE1, z2/count);
analogWrite(RED2, x/count);
analogWrite(GREEN2, y/count);
analogWrite(BLUE2, z/count);
count--;
delay(laggy);
}
}
void SOLID(uint8_t x, uint8_t y, uint8_t z, uint8_t x2, uint8_t y2, uint8_t z2)//solid color no fading.
{
analogWrite(RED2, x2);
analogWrite(GREEN2, y2);
analogWrite(BLUE2, z2);
analogWrite(RED1, x);
analogWrite(GREEN1, y);
analogWrite(BLUE1, z);
}
void BLINKY(uint8_t x, uint8_t y, uint8_t z, uint8_t x2, uint8_t y2, uint8_t z2)
{
uint16_t PULSE = 1500;
delay(PULSE);
analogWrite(RED2, x2);
analogWrite(GREEN2, y2);
analogWrite(BLUE2, z2);
analogWrite(RED1, x);
analogWrite(GREEN1, y);
analogWrite(BLUE1, z);
delay(PULSE);
analogWrite(RED2, x);
analogWrite(GREEN2, y);
analogWrite(BLUE2, z);
analogWrite(RED1, x2);
analogWrite(GREEN1, y2);
analogWrite(BLUE1, z2);
delay(PULSE);
}
uint8_t KeyInputHandler()
{
uint8_t Sig = 0;
if(Serial.available() > 0)
{
    Sig = Serial.read();
}
else
{
}
return(Sig);
void CleanUp()
{
    ANGH = 0;
    ANGT = 0;
    ANGO = 0;
    ANG = 0;
    Speed = 0;
    SpeedH = 0;
    SpeedT = 0;
    SpeedO = 0;
}

void Middleton_CNTRL()
{
    if(digitalRead(Controller) == 1)
        //LIGHTCONTROLLER();
    
    OPTION = KeyInputHandler();
    while(OPTION == 0)
    {
        if(state == 0)
            {
                Serial.println("*******MIDDLETON MOTOR CONTROLLER*******");
                Serial.println("*  *           *  *           *   *******");
                Serial.println("*  * *       * *  * *       * *   *");
                Serial.println("*  *   *   *   *  *   *   *   *  ");
                Serial.println("*  *     *     *  *     *     *   *******");
                Serial.println("*****************************************");
                Serial.println("*****************************************");
                Serial.println("For Motor Control Press s");
                Serial.println("For Color Press c");
                Serial.println("For Diagnostics Press d");
                Serial.println("*****************************************");
                CleanUp();
                LIGHTCONTROLLER();
                state++;
            }
    
    else
    {
        OPTION = KeyInputHandler();
    }
}
while(OPTION == 115)
{
    state = 0;
    while(state == 0)
    {
        if(wtt == 0)
        {
            Serial.println("Motor Control Entered!");
            Serial.println("");
            Serial.println("Please enter Speed");
            Serial.println("");
            wtt++;
        }
        else
        {
            if(Speed > 0)
            {
                Serial.println("Speed Selected ");
                Serial.println(Speed);
                Serial.println("");
                ANGH = 0;
                ANGT = 0;
                ANGO = 0;
                state++;
                wtt = 0;
            }
            else
            {
                if(SpeedH>0)
                {
                    if(SpeedT > 0)
                    {
                        if(SpeedO>0)
                        {
                            Speed = (SpeedH-48)*100 + (SpeedT-48)*10 + SpeedO-48;
                        }
                        else
                        {
                            SpeedO = KeyInputHandler();
                        }
                    }
                    else
                    {
                        SpeedT = KeyInputHandler();
                    }
                }
            }
        }
    }
}
else
{
    SpeedH = KeyInputHandler();
}

while(state == 1)
{
    if(wtt == 0)
    {
        Serial.println("Please enter Angle");
        Serial.println(" ");
        wtt++;
    }
    else
    {
        if(ANG > 0)
        {
            Serial.print("Angle Selected ");
            Serial.println(ANG);
            Serial.println(" ");
            state++;
            wtt = 0;
            ANGH = 0;
            ANGT = 0;
            ANGO = 0;
        }
    }
    if(ANG == 0)
    {
        if(ANGH>0)
        {
            if(ANGT > 0)
            {
                if(ANGO>0)
                {
                    ANG = (ANGH-48)*100 + (ANGT-48)*10 + ANGO-48;
                    if(ANG > MAX_Season_Angle)
                    {
                        ANG = MAX_Season_Angle;
                    }
                    else
                    {
                        ANGO = KeyInputHandler();
                    }
                }
            }
        }
    }
}
else
{
    ANGT = KeyInputHandler();
}
}
else
{
    ANGH = KeyInputHandler();
}
}

while(state == 2)
{
    if(wtt == 0)
    {
        Serial.print("Are you Sure you want to move to: ");
        Serial.print(ANG);
        Serial.print(" degrees at a speed of: ");
        Serial.println(Speed);
        Serial.println("yy or nn?");
        Serial.println("'\n");
        wtt++;
    }
    if(KeyInputHandler() == 121)
    {
        Serial.println("Moving Motor!");
        Serial.println("'\n");
        movemotor(ANG);
        OPTION = 0;
        state = 0;
        wtt = 0;
    }
    if(KeyInputHandler() == 110)
    {
        OPTION = 0;
        state = 0;
        wtt = 0;
    }
}
}
while(OPTION == 99)
{
    if(wtt == 0)
    {
Serial.println("*****************************************");
Serial.println("Color Controller Entered");
Serial.println(" Capital letters increas Color Intensity");
Serial.println(" R G B");
Serial.println(" Lowercase letters increas Color Intensity");
Serial.println(" r g b");
Serial.println(" Mode Options");
Serial.println(" f - fade");
Serial.println(" s - solid");
Serial.println(" x returns home");
Serial.println("*****************************************");

wtt++; }
else
{
 if(CIn == 114)
{
    DutyR = DutyR - 10;
    UpdateColor();
}
if(CIn == 98)
{
    DutyB = DutyB - 10;
    UpdateColor();
}
if(CIn == 103)
{
    DutyG = DutyG - 10;
    UpdateColor();
}
if(CIn == 82)
{
    DutyR = DutyR + 10;
    UpdateColor();
}
if(CIn == 66)
{
    DutyB = DutyB + 10;
    UpdateColor();
}
if(CIn == 71)
DutyG = DutyG + 10;
UpdateColor();
}
if(CIn == 102)
{
    Mode = 1;
    UpdateColor();
}
if(CIn == 115)
{
    Mode = 0;
    UpdateColor();
}
if(CIn == 120)
{
    OPTION = 0;
}
else
{
    CIn = KeyInputHandler();
}
}
while(OPTION == 100)
{
    Serial.println("*****************************************");
    Serial.print("Amount of recoveries: ");
    Serial.println(LimitThrow);
    Serial.print("DTANGLE: ");
    Serial.println(DTANGLE);
    Serial.print("Solar Panel Output Voltage: ");
    Serial.println(SCALEDVOLT);
    Serial.print("DTCOUNTER Value: ");
    Serial.println(DTCOUNTER);
    Serial.println("Press x to exit");
    Serial.println("*****************************************");
    delay(1000);
    if(CIn == 120)
    {
        OPTION = 0;
    }
    else
    {
        CIn = KeyInputHandler();
    }
}
void UpdateColor()
{
  Serial.println("STATUS");
  Serial.print("Red: ");
  Serial.println(DutyR);
  Serial.print("Blue: ");
  Serial.println(DutyB);
  Serial.print("Green: ");
  Serial.println(DutyG);
  Serial.print("Mode: ");
  if(Mode == 1)
  {
    Serial.println("Fade");
    FADE(DutyR, DutyB, DutyG, DutyR, DutyB, DutyG);
  }
  if(Mode == 0)
  {
    Serial.println("Solid");
    SOLID(DutyR, DutyB, DutyG, DutyR, DutyB, DutyG);
  }
  Serial.println("\n");
}

float Middleton_Moving_Average(uint16_t x, uint8_t opto)
{
  uint32_t SUM = 0;
  uint16_t samples;
  if(opto > 1)
  {
    samples = 1000;
  }
  else
  {
    samples = 100;
  }
  uint16_t counter = samples;
  MMAx[0] = x;
  MMAy[0] = x;
  MMAz[0] = x;
  MMAcomp[0] = x;
  while(counter >= 1)
  {
    if(opto == 0)
    {
      MMAx[counter] = MMAx[counter-1];
    }
SUM = SUM + MMAx[counter];
}
if(opto == 1)
{
    MMAy[counter] = MMAy[counter - 1];
    SUM = SUM + MMAy[counter];
}
if(opto == 2)
{
    MMAz[counter] = MMAz[counter - 1];
    SUM = SUM + MMAz[counter];
}
if(opto == 3)
{
    MMAcomp[counter] = MMAcomp[counter - 1];
    SUM = SUM + MMAcomp[counter];
}
counter--;
}
float MMMA = (float)SUM/samples;
return MMMA;
Appendix 4: Tracking Function

Test Code:
#include <TimerOne.h>

// Global Variables
volatile uint16_t DTCOUNTER;
volatile uint16_t NIGHTHOURS;
volatile uint16_t DTSUNRISE;
volatile uint16_t DTSUNSET;
volatile uint16_t DAYCOUNTER;

// Assign Pin names
uint8_t right = 10;
uint8_t left = 9;
char comp = A1;
char TiltAngle = A0;

void setup()
{
    // Initialize the Inputs and Outputs
    pinMode(comp, INPUT);
    pinMode(TiltAngle, INPUT);
    pinMode(right, OUTPUT);
    pinMode(left, OUTPUT);
    Serial.begin(9600);
    Timer1.initialize(5000000); // set timer to 5 seconds
    Timer1.attachInterrupt(HourIsr); // tell the timer to go to ISR
}

int COMPVOLT()
{
    float VREADING = analogRead(comp);
    uint8_t SCALEDVOLT = VREADING*5/30;
    Serial.println(SCALEDVOLT);
    Serial.println("COMPVOLT");
    delay(1000);
    return SCALEDVOLT;
}

int DTANGLE()
{
    float AREADING = analogRead(TiltAngle);
    uint8_t SCALEDANG = AREADING*5/30;
    Serial.println(SCALEDANG);
    Serial.println("TiltANgle");
    delay(1000);
    return SCALEDANG;
}

void loop()
{
    // Serial.println(DTCOUNTER);
void HourIsr()
{
    DTCOUNTER=DTCOUNTER+1;
    NIGHTHOURS=NIGHTHOURS+1;
    DTSUNRISE=DTSUNRISE+1;
    DTSUNSET=DTSUNSET+1;
    DAYCOUNTER=DAYCOUNTER+1;
}

void Tracker()
{
    uint8_t NewAngle; //Create a variable for the desired panel angle
    uint8_t PVOLT = 0; // Create a variable to store the current pannel voltage in
    uint8_t FINETUNEANGLE;
    uint8_t FourTries = 0;
    Serial.println("Begin Tracking");
    delay(1000);
    NewAngle=DTANGLE()+10; //create a value for the desired position
    delay(1000);
    movemotor(NewAngle);
    Serial.println("10 Degree difference Achieved");
    delay(5000); //wait 5 seconds to get a panel voltage reading may change depending on ability to
detect change
    FINETUNEANGLE = DTANGLE(); //set the finetuneangle to current position, prepare to
    calibrate
    while(COMPVOLT()>=PVOLT && (FourTries<4))
    {
        PVOLT = COMPVOLT();
        Serial.print("PVOLT");
        Serial.print(PVOLT);
        Serial.println("In the While Loop");
        FINETUNEANGLE = FINETUNEANGLE + 10; // increment posiition 5 degrees
        movemotor(FINETUNEANGLE);
        delay(5000);
        Serial.println("FineTuning Angle");
        Serial.println("FourTries");
        Serial.println(FourTries);
        Serial.println(COMPVOLT());
        FourTries = FourTries+1; // we will only try and find optimal position 4 times before returning
to main
} Serial.println("END of Tracking");
}

void NIGHT()
{
    NIGHTHOURS=0;
    DTSUNSET = 0;
    while(NIGHTHOURS<2160)
    {
        DTCOUNTER=0;
    }
    movemotor(0);
    while((COMPVOLT()) < 50)
    {
    }
    DTSUNRISE = 0;
}

void movemotor(int NewAngle)
{
    uint8_t hyststop = NewAngle+3; //Create a safety factor to prevent oscillation
    // move motor until it reaches desired position
    while (DTANGLE()<NewAngle)
    {
        Serial.println("moving to new angle");
        digitalWrite(right,HIGH);
        digitalWrite(left,LOW);
    }
    while (DTANGLE()>hyststop)
    {
        digitalWrite(right,LOW);
        digitalWrite(left,HIGH);
    }
    digitalWrite(right,LOW);
    digitalWrite(left,LOW);
}

Serial Monitor Results
Part 1
Begin Tracking
TiltANgle 0
moving to new angle
TiltANgle 6
moving to new angle
TiltANgle 9
moving to new angle
TiltANgle 13
10 Degree difference Achieved
COMPVOLT 0
PVOLT 0
In the While Loop
TiltANgle 12
moving to new angle
TiltANgle 13
moving to new angle
TiltANgle 13
Moving to new angle
TiltANgle 16
moving to new angle
TiltANgle 19
moving to new angle
TiltANgle 23
FineTuning Angle
FourTries 0
COMPVOLT 6
PVOLT 6
In the While Loop
TiltANgle 23
TiltANgle 26
moving to new angle
TiltANgle 29
moving to new angle
TiltANgle 33
FineTuning Angle
FourTries 1
COMPVOLT 0
END of Tracking

**Part 2**
Begin Tracking
TiltANgle 0
moving to new angle
TiltANgle 3
moving to new angle
TiltANgle 6
moving to new angle
TiltANgle 9
moving to new angle
TiltANgle 13
10 Degree difference Achieved
COMPVOLT 0
COMPVOLT 6
PVOLT 6
In the While Loop
TiltANgle 12
moving to new angle
TiltANgle 13
moving to new angle
TiltANgle 19
moving to new angle
TiltANgle 23
FineTuning Angle
FourTries 0
COMPVOLT 9
PVOLT 9
In the While Loop
22
TiltANgle
moving to new angle
23
TiltANgle
moving to new angle
23
TiltANgle
moving to new angle
26
TiltANgle
moving to new angle
29
TiltANgle
moving to new angle
29
TiltANgle
moving to new angle
29
TiltANgle
moving to new angle
33
TiltANgle
33
TiltANgle
FineTuning Angle
FourTries 1
COMPVOLT 16
PVOLT16
In the While Loop
TiltANgle 33
moving to new angle
TiltANgle 36
moving to new angle
TiltANgle 40
moving to new angle
TiltANgle 43
FineTuning Angle
FourTries 2
COMPVOLT 22
PVOLT 22
In the While Loop
TiltANgle 43
moving to new angle
TiltANgle 46
TiltANgle 50
TiltANgle 53
FineTuning Angle
FourTries 3
COMPVOLT 32
END of Tracking
Begin Tracking
Appendix 5: Night Function

Test Code:
#include <TimerOne.h>
void LIGHTCONTROLLER();
volatile uint16_t DTCOUNTER;
volatile uint16_t NIGHTHOURS;
volatile uint16_t DTSUNRISE;
volatile uint16_t DTSUNSET;
volatile uint16_t DAYCOUNTER;
uint8_t GREEN1 = 11;
uint8_t right = 6;
uint8_t left = 5;
char comp = A5;
char TiltAngle = A4;
void setup()
{
    pinMode(comp, INPUT);
    pinMode(TiltAngle, INPUT);
    pinMode(right, OUTPUT);
    pinMode(left, OUTPUT);
    pinMode(GREEN1, OUTPUT);
    Serial.begin(9600);
    Timer1.initialize(5000000); // set timer to 5 seconds
    Timer1.attachInterrupt(HourIsr); // tell the timer to go to ISR
}
int COMPVOLT()
{
    float VREADING = analogRead(comp);
    uint8_t SCALEDVOLT = VREADING*5/30;
    Serial.println(SCALEDVOLT);
    Serial.println("COMPVOLT");
    delay(1000);
    return SCALEDVOLT;
}
int DTANGLE()
{
    float AREADING = analogRead(TiltAngle);
    uint8_t SCALEDANG = AREADING*5/30;
    Serial.println(SCALEDANG);
    Serial.println("TiltAngle");
    delay(1000);
    return SCALEDANG;
}
void loop()
{
delay(1000);
if (DTANGLE()>80 && COMPVOLT()<53)
{
    NIGHT();
}
}
void HourIsr()
{
    DTCOUNTER=DTCOUNTER+1;
    NIGHTHOURS=NIGHTHOURS+1;
    DTSUNRISE=DTSUNRISE+1;
    DTSUNSET=DTSUNSET+1;
    DAYCOUNTER=DAYCOUNTER+1;
}
void NIGHT()
{
    NIGHTHOURS=0;
    DTSUNRISE = 0;
    Serial.println("Sunset Detected");
    while(NIGHTHOURS<6){
        delay(1000);
        LIGHTCONTROLLER();
    }
    Serial.println("30 seconds passed reset for sunrise");
movemotor(0);
    DTSUNRISE = 0;
    while((COMPVOLT()) < 56){
        Serial.println("Wait for a voltage value of 56 from comparator");
        delay(1000);
        LIGHTCONTROLLER();
    }
    DTSUNRISE = 0;
    Serial.println("Sunrise Detected!");
}
void movemotor(int NewAngle){
    uint8_t hyststop = NewAngle+3; //Create a safety factor to prevent oscillation
    // move motor until it reaches desired position
    while (DTANGLE()<NewAngle)
    {
        digitalWrite(right,HIGH);
        digitalWrite(left,LOW);
    }
    while (DTANGLE()>hyststop){

digitalWrite(right, LOW);
digitalWrite(left, HIGH);
}
Serial.println("Returned to Starting Position!");
digitalWrite(right, LOW);
digitalWrite(left, LOW);
}
void LIGHTCONTROLLER()
{
   if (DTSUNRISE < 5)
   {
      Serial.println("Turn Lights On");
      delay(1000);
      digitalWrite(GREEN1, HIGH);
   }
   if (DTSUNRISE > 6)
   {
      Serial.println("TurnOFFLIGHTS");
      delay(1000);
      digitalWrite(GREEN1, LOW);
   }
}
Serial Monitor
TiltANgle 0
TiltANgle 69
TiltANgle 73
TiltANgle 79
TiltANgle 83
COMPVOLT 73
TiltANgle 86
COMPVOLT 63
TiltANgle 86
COMPVOLT 56
TiltANgle 86
COMPVOLT 52
Sunset Detected
Turn Lights On
30 seconds passed reset for sunrise
TiltANgle 87
TiltANgle 84
TiltANgle 83
TiltANgle 65
TiltANgle 48
TiltANgle 8
TiltANgle 0
Returned to Starting Position!
COMPVOLT 42
Wait for a voltage value of 56 from comparator
Turn Lights On
COMPVOLT 32
Wait for a voltage value of 56 from comparator
Turn Lights On
COMPVOLT 29
Wait for a voltage value of 56 from comparator
Turn Lights On
COMPVOLT 73
Sunrise Detected!
Appendix 6: External Interrupts for Light Control

Test Code:
uint8_t TOGGLE1 = 2;
uint8_t TOGGLE2 = 3;
uint8_t TOGGLE3 = 0;
void SVAL();
void setup() {
    pinMode(TOGGLE1, INPUT);
    pinMode(TOGGLE2, INPUT);
    pinMode(TOGGLE3, INPUT);
    attachInterrupt(digitalPinToInterrupt(TOGGLE1), SVAL, CHANGE); // Change the Switchval variable on change
    attachInterrupt(digitalPinToInterrupt(TOGGLE2), SVAL, CHANGE); // Change the Switchval variable on change
    Serial.begin(9600);
}
void loop() {
}
void SVAL() {
    // create a three bit value to represent the 3 toggle switches used for user input.
    Serial.println("enterhandler ");
    Serial.print("NEW VALUE ");
    uint8_t LIGHTCONFIG = (digitalRead(TOGGLE2))<<1 | (digitalRead(TOGGLE1)) |
    (digitalRead(TOGGLE3))<<2; // create a 3 bit lightconfig value
    Serial.println(LIGHTCONFIG);
    delay(2000);
}

Serial Monitor Results
enterhandler
NEW VALUE  0
enterhandler
NEW VALUE  1
enterhandler
NEW VALUE  2
enterhandler
NEW VALUE  3
enterhandler
NEW VALUE  4
enterhandler
NEW VALUE  5
enterhandler
NEW VALUE  6
enterhandler
NEW VALUE  7
#include <DueTimer.h>
float COMP_Voltage[1000];
float MMAcomp[100];
float RAW_Voltage[1000];
float Middleton_Moving_Average(float x, uint8_t opto);
char Comparator = A0;
volatile float SCALEDVOLT;
volatile uint16_t MINUTE;
volatile uint8_t TKEEP;
volatile float VIN;
volatile float SUM;
void setup()
{
  pinMode(Comparator, INPUT);
  Timer1.attachInterrupt(record_share); // tell the timer to go to ISR
  Timer1.start(5000000); // set timer to 5 seconds
  Timer2.attachInterrupt(COMPVOLT);
  Timer2.start(100000); // check every 100ms
  Serial.begin(9600);
  Serial.print("Minute ");
  Serial.print("Voltage ");
  Serial.println("NONFILTER V");
}
void loop()
{
}

void record_share()
{
  if(TKEEP >= 5)
  {
    MINUTE++;
    Serial.print(MINUTE);
    Serial.print(" ");
    Serial.print(SCALEDVOLT);
    Serial.print(" ");
    Serial.println(VIN);
    RAW_Voltage[MINUTE] = VIN;
    COMP_Voltage[MINUTE] = SCALEDVOLT;
    TKEEP = 0;
  }
  else
  {
    
  
}
TKEEP++;
}

void COMPVOLT()
{
    float VREADING = analogRead(Comparator);
    VIN = VREADING*3.3/1023*(3.147/.47)*(13.46/12.7); //a value between 0 and 22
    SCALEDVOLT = Middleton_Moving_Average(VIN, 3);
}

float Middleton_Moving_Average(float x, uint8_t opto)
{
    SUM=0;
    float samples;
    samples = 25;
    uint16_t counter = samples;
    MMAcomp[0] = x;
    while(counter > 0)
    {
        MMAcomp[counter] = MMAcomp[counter - 1];
        SUM = SUM + MMAcomp[counter];
        counter--;
    }
    float MMMA = SUM/(samples);
    return MMMA;
}
Appendix 8: Move Motor Script

char xPin = A1;
char yPin = A2;
char zPin = A3;
uint8_t Dir = 5;
uint8_t Brake = 4;
uint8_t SpeedSelect = 3;
volatile uint8_t xAng;
volatile uint8_t yAng;
volatile uint8_t zAng;
volatile uint8_t Speed;
volatile uint8_t SCALEDVOLT;
volatile uint8_t DTANGLE;
char Comparator = A0;

void setup()
{
    pinMode(Dir, OUTPUT);
    pinMode(Brake, OUTPUT);
    pinMode(SpeedSelect, OUTPUT);
    pinMode(xPin, INPUT);
    pinMode(yPin, INPUT);
    pinMode(zPin, INPUT);
    pinMode(Comparator, INPUT);
    Speed = 25;
    DTANGLE = xAng;
}

void loop()
{
    movemotor(5);
    movemotor(165);
    Speed = Speed + 25;
}
void COMPVOLT()
{
    float VREADING = analogRead(Comparator);
    SCALEDVOLT = VREADING*100/3.3;
}
void ANGLE()
{
    uint8_t minValx = 299;
    uint8_t maxValx = 450;
    uint8_t minValy = 296;
    uint8_t maxValy = 447;
    uint8_t minValz = 303;
    uint8_t maxValz = 453;
//read the analog values from the accelerometer
uint8_t xRead = analogRead(xPin);
uint8_t yRead = analogRead(yPin);
uint8_t zRead = analogRead(zPin);
//convert read values to degrees -90 to 90 - Needed for atan2
uint8_t xAng = map(xRead, minValx, maxValx, -90, 90);
uint8_t yAng = map(yRead, minValy, maxValy, -90, 90);
uint8_t zAng = map(zRead, minValz, maxValz, -90, 90);
if(yAng>0)
{
    zAng=180-zAng;
}
Serial.print("xAng: ");
Serial.print(xAng);
Serial.print(" | yAng: ");
Serial.print(yAng);
Serial.print(" | zAng: ");
Serial.println(zAng);
}
void movemotor(uint8_t NEWANGLE)
{
    if(abs(DTANGLE - NEWANGLE)>3)
    {
        while(DTANGLE < NEWANGLE)
        {
            digitalWrite(Dir, HIGH); // set direction
            analogWrite(SpeedSelect, Speed); // set speed
            digitalWrite(Brake, LOW); // release brake
            analogWrite(SpeedSelect, 0); // reduce the speed to 0;
            digitalWrite(Brake, HIGH); //Apply the brake;
            delay(50);
            digitalWrite(Brake, LOW);
        }
        while(DTANGLE > NEWANGLE)
        {
            digitalWrite(Dir, LOW); // set direction
            analogWrite(SpeedSelect, Speed); // set speed
            digitalWrite(Brake, LOW); // release brake
        }
        analogWrite(SpeedSelect, 0); // reduce the speed to 0;
        digitalWrite(Brake, HIGH); //Apply the brake;
    }
    else
    {
    }
}
### Appendix 9: Functions Variables and PinMap

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HourIsr</strong></td>
<td>Increment DTCOUNTER every 5 seconds. This program also increments NIGHTHOURS counter every 5 seconds.</td>
</tr>
<tr>
<td><strong>Tracker</strong></td>
<td>This function moves creates a desired angle for the solar panel to move too. The function uses a while loop to check the voltage at 4 different positions. If the voltage drops from one position to the next, the solar panel will stop tracking the sun.</td>
</tr>
<tr>
<td><strong>movemotor</strong></td>
<td>This function receives a desired input angle and then moves the motor in the direction of that angle until it reaches the desired position. When the solar panel is in the desired position the program returns to main.</td>
</tr>
<tr>
<td><strong>LIGHTCONTROLLER</strong></td>
<td>This function takes the state of the toggle switches and changes the RGB LED's accordingly. This function calls the SOLID, FADE, and BLINKY functions.</td>
</tr>
<tr>
<td><strong>TURNOFF</strong></td>
<td>This function is used to stop the panel from tracking the sun. When the AllOff switch is set high the algorithm will be stuck in an infinite while loop.</td>
</tr>
<tr>
<td><strong>Recovery</strong></td>
<td>This function is triggered when the limit switches are set high. This function autonomously recovers the system when the motor moves the solar panel outside of its designed range.</td>
</tr>
<tr>
<td><strong>SVAL</strong></td>
<td>The switch value function SVAL takes the state of the user controlled toggle switches and alters the lighting display based on the input. Returns a two bit value for lighting control.</td>
</tr>
<tr>
<td><strong>ANGLE</strong></td>
<td>This function returns the solar panel position. This function also detects the current season based on the tilt of the solar panel. The maximum range of rotation is set in this program as well.</td>
</tr>
<tr>
<td><strong>COMPVOLT</strong></td>
<td>This function reads the value from the comparator circuit and returns a scaled value between 0 and 100.</td>
</tr>
<tr>
<td><strong>SOLID</strong></td>
<td>A function called by the lightcontroller that sets the LEDs at a constant color.</td>
</tr>
<tr>
<td><strong>BLINKY</strong></td>
<td>A function called by the lightcontroller to blink the LED’s on and off.</td>
</tr>
<tr>
<td><strong>FADE</strong></td>
<td>A function called by the lightcontroller to fade the LED's between colors.</td>
</tr>
</tbody>
</table>
KeyInputHandler This function is used by the remote controller to read from the user's laptop.

CleanUp This function is used by the remote controller to clear previously read values.

Middleton_CNTRL This function provides the user with a software remote control. The user communicates with the Arduino® using their laptop keyboard. The motor and RGB Led's can be controlled and a diagnostics page is available as well.

Middleton_Moving_Average This function is used to stabilize the incoming data from the solar panel voltage and accelerometer.

NIGHT This function is called when panel output voltage falls below a threshold level and when the solar panel position is greater than 135°. This program waits for the NIGHTHOURS counter to reach 2160 (holding the final position for 3 hours during sunset). The DTCOUNTER is reset to prevent any errors in the logic. After waiting three hours the motor is moved to sunrise angle. the program then waits for the comparator voltage to reach threshold sunrise levels to return to the main program.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Max value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT COUNTER</td>
<td>Volatile uint16_t</td>
<td>720</td>
<td>Increment every 5 seconds, when it reaches 720 an hour has passed</td>
</tr>
<tr>
<td>DT ANGLE</td>
<td>Volatile uint8_t</td>
<td>180</td>
<td>Store the value for the tilt angle sensor</td>
</tr>
<tr>
<td>VREADING</td>
<td>float</td>
<td>3.3</td>
<td>Store the voltage value of the comparator circuit</td>
</tr>
<tr>
<td>SCALEDVOLT</td>
<td>float</td>
<td>100</td>
<td>Scales the value of the comparator circuit between 0 and 100.</td>
</tr>
<tr>
<td>NewAngle</td>
<td>uint8_t</td>
<td>180</td>
<td>Desired solar panel angle 10 degrees higher than current angle.</td>
</tr>
<tr>
<td>PVOLT</td>
<td>uint8_t</td>
<td>250</td>
<td>Variable created to compare the comparator voltage from position to position.</td>
</tr>
<tr>
<td>FINETUNEANGLE</td>
<td>uint8_t</td>
<td>180</td>
<td>Variable used to increment the desired solar panel position 5 degrees</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
<td>-------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FourTries</td>
<td>uint8_t</td>
<td>2</td>
<td>Counter used to ensure that the solar panel does not try more than 2 different positions while finding optimal position.</td>
</tr>
<tr>
<td>NIGHTHOURS</td>
<td>Volatile uint16_t</td>
<td>2160</td>
<td>NIGHTHOURS is a counter used to calculate when 3 hours have passed.</td>
</tr>
<tr>
<td>DTSUNRISE</td>
<td>Volatile uint16_t</td>
<td>15840</td>
<td>This counter is incremented every 5 seconds for 22 hours. This counter begins counting when the solar panel exits Night mode.</td>
</tr>
<tr>
<td>DTSUNSET</td>
<td>Volatile uint16_t</td>
<td>1440</td>
<td>This counter is incremented every 5 seconds for 2 hours and begins counting when sunset is detected.</td>
</tr>
<tr>
<td>MOTIME</td>
<td>float</td>
<td>9900</td>
<td>Value representing the amount of time to run the motor.</td>
</tr>
<tr>
<td>count</td>
<td>uint8_t</td>
<td>254</td>
<td>Value used to fade the LED's on and off.</td>
</tr>
<tr>
<td>LIGHTCONFIG</td>
<td>uint8_t</td>
<td>7</td>
<td>Lighting configuration setting based on the value from the input switches. This value is derived by shifting toggle1 and toggle 2 to the left and oring it with toggle 3 creating a 3 bit binary value</td>
</tr>
<tr>
<td>xAng</td>
<td>Volatile float</td>
<td>180</td>
<td>Read the angle in X direction.</td>
</tr>
<tr>
<td>yAng</td>
<td>Volatile float</td>
<td>180</td>
<td>read the angle in Y direction.</td>
</tr>
<tr>
<td>zAng</td>
<td>Volatile float</td>
<td>180</td>
<td>read the angle in the Z direction.</td>
</tr>
<tr>
<td>LimitThrow</td>
<td>volatile uint8_t</td>
<td>255</td>
<td>record the amount of recoveries</td>
</tr>
<tr>
<td>MAX_Season_Angle</td>
<td>volatile uint8_t</td>
<td>255</td>
<td>set the value of the maximum angle of rotation depending on the current season.</td>
</tr>
<tr>
<td>Speed</td>
<td>volatile uint8_t</td>
<td>255</td>
<td>global variable used for setting the duty cycle of the motor</td>
</tr>
</tbody>
</table>
### Array

<table>
<thead>
<tr>
<th>MMAx</th>
<th>Array</th>
<th>100 elements</th>
<th>store the last 100 accelerometer values read in the x direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMAy</td>
<td>Array</td>
<td>100 elements</td>
<td>store the last 100 accelerometer values read in the y direction</td>
</tr>
<tr>
<td>MMAz</td>
<td>Array</td>
<td>100 elements</td>
<td>store the last 100 accelerometer values read in the z direction</td>
</tr>
<tr>
<td>MMAcomp</td>
<td>Array</td>
<td>100 elements</td>
<td>store the last 100 voltage readings from the solar panel</td>
</tr>
</tbody>
</table>

### PinMap

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Pin Number</th>
<th>Input/Output</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllOff</td>
<td>53</td>
<td>Input</td>
<td>Turn off lights and motor.</td>
</tr>
<tr>
<td>Controller</td>
<td>52</td>
<td>Input</td>
<td>Enter color and motor controller, along with diagnostics.</td>
</tr>
<tr>
<td>RED1</td>
<td>12</td>
<td>Output</td>
<td>Red Strand 1</td>
</tr>
<tr>
<td>GREEN1</td>
<td>11</td>
<td>Output</td>
<td>Green Strand 1</td>
</tr>
<tr>
<td>BLUE1</td>
<td>10</td>
<td>Output</td>
<td>Blue Strand 1</td>
</tr>
<tr>
<td>RED2</td>
<td>8</td>
<td>Output</td>
<td>Red Strand 2</td>
</tr>
<tr>
<td>GREEN2</td>
<td>7</td>
<td>Output</td>
<td>Green Strand 2</td>
</tr>
<tr>
<td>BLUE2</td>
<td>6</td>
<td>Output</td>
<td>Blue Strand 2</td>
</tr>
<tr>
<td>TOGGLE1</td>
<td>32</td>
<td>Input</td>
<td>User input for lighting control</td>
</tr>
<tr>
<td>TOGGLE2</td>
<td>34</td>
<td>Input</td>
<td>User input for lighting control</td>
</tr>
<tr>
<td>TOGGLE3</td>
<td>36</td>
<td>Input</td>
<td>User input for lighting control</td>
</tr>
<tr>
<td>Comparator</td>
<td>A0</td>
<td>Input</td>
<td>receive input from solar panel voltage</td>
</tr>
<tr>
<td>xPin</td>
<td>A1</td>
<td>Input</td>
<td>x position</td>
</tr>
<tr>
<td>yPin</td>
<td>A2</td>
<td>Input</td>
<td>y position</td>
</tr>
<tr>
<td>zPin</td>
<td>A3</td>
<td>Input</td>
<td>z position</td>
</tr>
<tr>
<td>KILLSWITCH1</td>
<td>46</td>
<td>Input</td>
<td>Read from limit switch 1</td>
</tr>
<tr>
<td>KILLSWITCH2</td>
<td>47</td>
<td>Input</td>
<td>Read from limit switch 2</td>
</tr>
<tr>
<td>Dir</td>
<td>5</td>
<td>Output</td>
<td>Enable the H-Bridge to turn left/right</td>
</tr>
<tr>
<td>Brake</td>
<td>4</td>
<td>Output</td>
<td>Stop the Motor</td>
</tr>
<tr>
<td>SpeedSelect</td>
<td>3</td>
<td>Output</td>
<td>Control the speed of motor</td>
</tr>
</tbody>
</table>